

"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



For training, visit [mindshare.com](http://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

<b>Intel Architecture</b> <ul style="list-style-type: none"><li>• Intel Ivy Bridge Processor</li><li>• Intel 64 (x86) Architecture</li><li>• Intel QuickPath Interconnect (QPI)</li><li>• Computer Architecture</li></ul>	<b>Virtualization Technology</b> <ul style="list-style-type: none"><li>• PC Virtualization</li><li>• IO Virtualization</li></ul>
<b>AMD Architecture</b> <ul style="list-style-type: none"><li>• AMD Opteron Processor (Bulldozer)</li><li>• AMD64 Architecture</li></ul>	<b>IO Buses</b> <ul style="list-style-type: none"><li>• PCI Express 3.0</li><li>• USB 3.0 / 2.0</li><li>• xHCI for USB</li></ul>
<b>Firmware Technology</b> <ul style="list-style-type: none"><li>• UEFI Architecture</li><li>• BIOS Essentials</li></ul>	<b>Storage Technology</b> <ul style="list-style-type: none"><li>• SAS Architecture</li><li>• Serial ATA Architecture</li><li>• NVMe Architecture</li></ul>
<b>ARM Architecture</b> <ul style="list-style-type: none"><li>• ARM Architecture</li></ul>	<b>Memory Technology</b> <ul style="list-style-type: none"><li>• Modern DRAM Architecture</li></ul>
<b>Graphics Architecture</b> <ul style="list-style-type: none"><li>• Graphics Hardware Architecture</li></ul>	<b>High Speed Design</b> <ul style="list-style-type: none"><li>• High Speed Design</li><li>• EMI/EMC</li></ul>
<b>Programming</b> <ul style="list-style-type: none"><li>• X86 Architecture Programming</li><li>• X86 Assembly Language Basics</li><li>• OpenCL Programming</li></ul>	<b>Surface-Mount Technology (SMT)</b> <ul style="list-style-type: none"><li>• SMT Manufacturing</li><li>• SMT Testing</li></ul>

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.

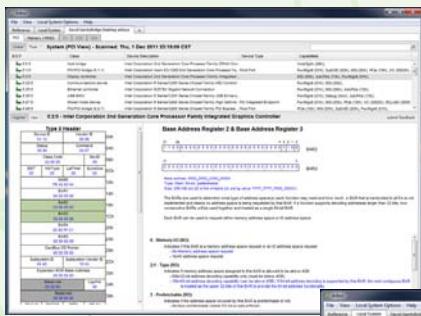
[training@mindshare.com](mailto:training@mindshare.com)

**1-800-633-1440**

[www.mindshare.com](http://www.mindshare.com)



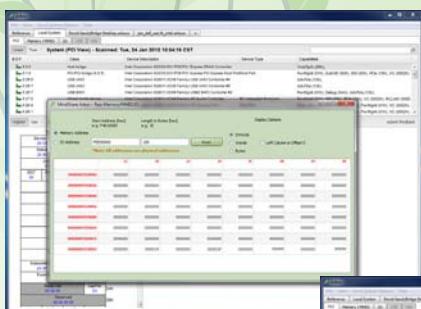
# The Ultimate Tool to View, Edit and Verify Configuration Settings of a Computer



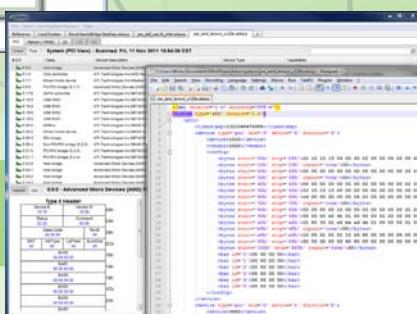
## Decode Data from Live Systems



## 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.)



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.

# *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*

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designators appear in this book, and MindShare was aware of the trademark claim, the designations have been printed in initial capital letters or all capital letters.

The authors and publishers have taken care in preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

*Library of Congress Cataloging-in-Publication Data*

Jackson, Mike and Budruk, Ravi

PCI Express Technology / MindShare, Inc., Mike Jackson, Ravi Budruk....[et al.]

Includes index

ISBN: 978-0-9836465-2-5 (alk. paper)

1. Computer Architecture. 2.0 Microcomputers - buses.

I. Jackson, Mike II. MindShare, Inc. III. Title

Library of Congress Number: 2011921066

ISBN: 978-0-9836465-2-5

Copyright ©2012 by MindShare, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.  
Printed in the United States of America.

Editors: Joe Winkles and Don Anderson

Project Manager: Maryanne Daves

Cover Design: Greenhouse Creative and MindShare, Inc.

Set in 10 point Palatino Linotype by MindShare, Inc.

Text printed on recycled and acid-free paper

First Edition, First Printing, September, 2012

"This book is dedicated to my sons, Jeremy and Bryan – I love you guys deeply. Creating a book takes a long time and a team effort, but it's finally done and now you hold the results in your hand. It's a picture of the way life is sometimes: investing over a long time with your team before you see the result. You were a gift to us when you were born and we've invested in you for many years, along with a number of people who have helped us. Now you've become fine young men in your own right and it's been a joy to become your friend as grown men. What will you invest in that will become the big achievements in your lives? I can hardly wait to find out."



# Acknowledgments

Thanks to those who made significant contributions to this book:

Maryanne Daves - for being book project manager and getting the book to press in a timely manner.

Don Anderson - for excellent work editing numerous chapters and doing a complete re-write of Chapter 8 on "Transaction Ordering".

Joe Winkles - for his superb job of technical editing and doing a complete re-write of Chapter 4 on "Address Space and Transaction Routing".

Jay Trodden - for his contribution in developing Chapter 4 on "Address Space and Transaction Routing"

Special thanks to LeCroy Corporation, Inc. for supplying:

*Appendix A: Debugging PCI Express™ Traffic using LeCroy Tools*

Special thanks to PLX Technology for contributing two appendices:

*Appendix B: Markets & Applications for PCI Express™*

*Appendix C: Implementing Intelligent Adapters and Multi-Host Systems  
With PCI Express™ Technology*

Thanks also to the PCI SIG for giving permission to use some of the mechanical drawings from the specification.

## Revision Updates:

1.0 - Initial eBook release

1.01 - Fixed Revision ID field in Figures 1-12, 1-13, 4-2, 4-4, 4-5, 4-6, 4-8, 4-9, 4-10, 4-17, 4-20, 4-21

---

# Contents

---

---

## About This Book

The MindShare Technology Series .....	1
Cautionary Note .....	2
Intended Audience .....	2
Prerequisite Knowledge .....	2
Book Topics and Organization .....	3
Documentation Conventions .....	3
PCI Express™ .....	3
Hexadecimal Notation .....	4
Binary Notation .....	4
Decimal Notation .....	4
Bits, Bytes and Transfers Notation .....	4
Bit Fields .....	4
Active Signal States.....	5
Visit Our Web Site.....	5
We Want Your Feedback.....	5

---

## Part One: The Big Picture

---

### Chapter 1: Background

Introduction.....	9
PCI and PCI-X .....	10
PCI Basics .....	11
Basics of a PCI-Based System .....	11
PCI Bus Initiator and Target.....	12
Typical PCI Bus Cycle .....	13
Reflected-Wave Signaling.....	16
PCI Bus Architecture Perspective .....	18
PCI Transaction Models .....	18
Programmed I/O .....	18
Direct Memory Access (DMA).....	19
Peer-to-Peer .....	20
PCI Bus Arbitration .....	20
PCI Inefficiencies.....	21
PCI Retry Protocol .....	21
PCI Disconnect Protocol .....	22
PCI Interrupt Handling.....	23
PCI Error Handling.....	24
PCI Address Space Map.....	25
PCI Configuration Cycle Generation .....	26

---

# Contents

---

PCI Function Configuration Register Space .....	27
Higher-bandwidth PCI .....	29
Limitations of 66 MHz PCI bus .....	30
Signal Timing Problems with the Parallel PCI Bus Model beyond 66 MHz.....	31
<b>Introducing PCI-X.....</b>	<b>31</b>
PCI-X System Example.....	31
PCI-X Transactions .....	32
PCI-X Features.....	33
Split-Transaction Model.....	33
Message Signaled Interrupts.....	34
Transaction Attributes .....	35
No Snoop (NS): .....	35
Relaxed Ordering (RO): .....	35
Higher Bandwidth PCI-X.....	36
Problems with the Common Clock Approach of PCI and PCI-X 1.0	
Parallel Bus Model .....	36
PCI-X 2.0 Source-Synchronous Model.....	37

---

## Chapter 2: PCIe Architecture Overview

<b>Introduction to PCI Express .....</b>	<b>39</b>
Software Backward Compatibility .....	41
Serial Transport.....	41
The Need for Speed .....	41
Overcoming Problems .....	41
Bandwidth .....	42
PCIe Bandwidth Calculation.....	43
Differential Signals .....	44
No Common Clock .....	45
Packet-based Protocol .....	46
Links and Lanes .....	46
Scalable Performance .....	46
Flexible Topology Options .....	47
Some Definitions .....	47
Root Complex.....	48
Switches and Bridges .....	48
Native PCIe Endpoints and Legacy PCIe Endpoints .....	49
Software Compatibility Characteristics.....	49
System Examples .....	52
<b>Introduction to Device Layers .....</b>	<b>54</b>
Device Core / Software Layer .....	59
Transaction Layer.....	59
TLP (Transaction Layer Packet) Basics.....	60

# Contents

---

TLP Packet Assembly.....	62
TLP Packet Disassembly .....	64
Non-Posted Transactions.....	65
Ordinary Reads.....	65
Locked Reads .....	66
IO and Configuration Writes .....	68
Posted Writes.....	69
Memory Writes .....	69
Message Writes .....	70
Transaction Ordering .....	71
Data Link Layer.....	72
DLLPs (Data Link Layer Packets) .....	73
DLLP Assembly .....	73
DLLP Disassembly .....	73
Ack/Nak Protocol .....	74
Flow Control.....	76
Power Management.....	76
Physical Layer.....	76
General .....	76
Physical Layer - Logical .....	77
Link Training and Initialization .....	78
Physical Layer - Electrical.....	78
Ordered Sets .....	79
<b>Protocol Review Example .....</b>	<b>81</b>
Memory Read Request.....	81
Completion with Data.....	83

---

## Chapter 3: Configuration Overview

<b>Definition of Bus, Device and Function.....</b>	<b>85</b>
PCIe Buses.....	86
PCIe Devices .....	86
PCIe Functions.....	86
<b>Configuration Address Space .....</b>	<b>88</b>
PCI-Compatible Space.....	88
Extended Configuration Space .....	89
<b>Host-to-PCI Bridge Configuration Registers.....</b>	<b>90</b>
General.....	90
Only the Root Sends Configuration Requests .....	91
<b>Generating Configuration Transactions.....</b>	<b>91</b>
Legacy PCI Mechanism.....	91
Configuration Address Port.....	92
Bus Compare and Data Port Usage.....	93

# Contents

---

Single Host System .....	94
Multi-Host System.....	96
Enhanced Configuration Access Mechanism .....	96
General .....	96
Some Rules.....	98
<b>Configuration Requests .....</b>	<b>99</b>
Type 0 Configuration Request .....	99
Type 1 Configuration Request .....	100
<b>Example PCI-Compatible Configuration Access .....</b>	<b>102</b>
<b>Example Enhanced Configuration Access.....</b>	<b>103</b>
<b>Enumeration - Discovering the Topology .....</b>	<b>104</b>
Discovering the Presence or Absence of a Function .....	105
Device not Present .....	105
Device not Ready .....	106
Determining if a Function is an Endpoint or Bridge .....	108
<b>Single Root Enumeration Example.....</b>	<b>109</b>
<b>Multi-Root Enumeration Example.....</b>	<b>114</b>
General.....	114
Multi-Root Enumeration Process.....	114
<b>Hot-Plug Considerations .....</b>	<b>116</b>
<b>MindShare Arbor: Debug/Validation/Analysis and Learning Software Tool.....</b>	<b>117</b>
General.....	117
MindShare Arbor Feature List .....	119

---

## Chapter 4: Address Space & Transaction Routing

<b>I Need An Address.....</b>	<b>121</b>
Configuration Space .....	122
Memory and IO Address Spaces .....	122
General .....	122
Prefetchable vs. Non-prefetchable Memory Space .....	123
<b>Base Address Registers (BARs) .....</b>	<b>126</b>
General.....	126
BAR Example 1: 32-bit Memory Address Space Request .....	128
BAR Example 2: 64-bit Memory Address Space Request .....	130
BAR Example 3: IO Address Space Request .....	133
All BARs Must Be Evaluated Sequentially.....	135
Resizable BARs.....	135
<b>Base and Limit Registers .....</b>	<b>136</b>
General.....	136
Prefetchable Range (P-MMIO) .....	137
Non-Prefetchable Range (NP-MMIO).....	139
IO Range .....	141

# Contents

---

Unused Base and Limit Registers .....	144
<b>Sanity Check: Registers Used For Address Routing .....</b>	<b>144</b>
<b>TLP Routing Basics .....</b>	<b>145</b>
Receivers Check For Three Types of Traffic .....	147
Routing Elements .....	147
Three Methods of TLP Routing .....	147
General .....	147
Purpose of Implicit Routing and Messages .....	148
Why Messages? .....	148
How Implicit Routing Helps .....	148
Split Transaction Protocol .....	149
Posted versus Non-Posted .....	150
Header Fields Define Packet Format and Type .....	151
General .....	151
Header Format/Type Field Encodings .....	153
TLP Header Overview .....	154
<b>Applying Routing Mechanisms .....</b>	<b>155</b>
ID Routing .....	155
Bus Number, Device Number, Function Number Limits .....	155
Key TLP Header Fields in ID Routing .....	155
Endpoints: One Check .....	156
Switches (Bridges): Two Checks Per Port .....	157
Address Routing .....	158
Key TLP Header Fields in Address Routing .....	159
TLPs with 32-Bit Address .....	159
TLPs with 64-Bit Address .....	159
Endpoint Address Checking .....	160
Switch Routing .....	161
Downstream Traveling TLPs (Received on Primary Interface) .....	162
Upstream Traveling TLPs (Received on Secondary Interface) .....	163
Multicast Capabilities .....	163
Implicit Routing .....	163
Only for Messages .....	163
Key TLP Header Fields in Implicit Routing .....	164
Message Type Field Summary .....	164
Endpoint Handling .....	165
Switch Handling .....	165
<b>DLLPs and Ordered Sets Are Not Routed .....</b>	<b>166</b>

# Contents

---

## Part Two: Transaction Layer

---

### Chapter 5: TLP Elements

Introduction to Packet-Based Protocol.....	169
General.....	169
Motivation for a Packet-Based Protocol .....	171
1. Packet Formats Are Well Defined .....	171
2. Framing Symbols Define Packet Boundaries.....	171
3. CRC Protects Entire Packet .....	172
Transaction Layer Packet (TLP) Details.....	172
TLP Assembly And Disassembly .....	172
TLP Structure.....	174
Generic TLP Header Format .....	175
General .....	175
Generic Header Field Summary .....	175
Generic Header Field Details .....	178
Header Type/Format Field Encodings .....	179
Digest / ECRC Field.....	180
ECRC Generation and Checking .....	180
Who Checks ECRC? .....	180
Using Byte Enables .....	181
General .....	181
Byte Enable Rules .....	181
Byte Enable Example.....	182
Transaction Descriptor Fields .....	182
Transaction ID .....	183
Traffic Class .....	183
Transaction Attributes .....	183
Additional Rules For TLPs With Data Payloads.....	183
Specific TLP Formats: Request & Completion TLPs.....	184
IO Requests.....	184
IO Request Header Format .....	185
IO Request Header Fields.....	186
Memory Requests .....	188
Memory Request Header Fields.....	188
Memory Request Notes .....	192
Configuration Requests .....	192
Definitions Of Configuration Request Header Fields .....	193
Configuration Request Notes .....	196

# Contents

---

Completions.....	196
Definitions Of Completion Header Fields .....	197
Summary of Completion Status Codes .....	200
Calculating The Lower Address Field.....	200
Using The Byte Count Modified Bit.....	201
Data Returned For Read Requests: .....	201
Receiver Completion Handling Rules:.....	202
Message Requests .....	203
Message Request Header Fields.....	204
Message Notes: .....	206
INTx Interrupt Messages.....	206
Power Management Messages .....	208
Error Messages.....	209
Locked Transaction Support.....	209
Set Slot Power Limit Message.....	210
Vendor-Defined Message 0 and 1 .....	210
Ignored Messages .....	211
Latency Tolerance Reporting Message.....	212
Optimized Buffer Flush and Fill Messages.....	213

---

## Chapter 6: Flow Control

Flow Control Concept .....	215
Flow Control Buffers and Credits.....	217
VC Flow Control Buffer Organization.....	218
Flow Control Credits .....	219
Initial Flow Control Advertisement .....	219
Minimum and Maximum Flow Control Advertisement .....	219
Infinite Credits.....	221
Special Use for Infinite Credit Advertisements.....	221
Flow Control Initialization.....	222
General.....	222
The FC Initialization Sequence.....	223
FC_Init1 Details .....	224
FC_Init2 Details .....	225
Rate of FC_INIT1 and FC_INIT2 Transmission .....	226
Violations of the Flow Control Initialization Protocol .....	227
Introduction to the Flow Control Mechanism.....	227
General.....	227
The Flow Control Elements .....	227
Transmitter Elements .....	228
Receiver Elements.....	229

# Contents

---

<b>Flow Control Example .....</b>	<b>230</b>
Stage 1 — Flow Control Following Initialization.....	230
Stage 2 — Flow Control Buffer Fills Up.....	233
Stage 3 — Counters Roll Over.....	234
Stage 4 — FC Buffer Overflow Error Check .....	235
<b>Flow Control Updates .....</b>	<b>237</b>
FC_Update DLLP Format and Content.....	238
Flow Control Update Frequency .....	239
Immediate Notification of Credits Allocated .....	239
Maximum Latency Between Update Flow Control DLLPs.....	240
Calculating Update Frequency Based on Payload Size and Link Width .....	240
Error Detection Timer — A Pseudo Requirement .....	243

---

## Chapter 7: Quality of Service

<b>Motivation .....</b>	<b>245</b>
<b>Basic Elements .....</b>	<b>246</b>
Traffic Class (TC).....	247
Virtual Channels (VCs) .....	247
Assigning TCs to each VC — TC/VC Mapping .....	248
Determining the Number of VCs to be Used .....	249
Assigning VC Numbers (IDs) .....	251
<b>VC Arbitration .....</b>	<b>252</b>
General.....	252
Strict Priority VC Arbitration .....	253
Group Arbitration.....	255
Hardware Fixed Arbitration Scheme .....	257
Weighted Round Robin Arbitration Scheme.....	257
Setting up the Virtual Channel Arbitration Table .....	258
<b>Port Arbitration .....</b>	<b>261</b>
General.....	261
Port Arbitration Mechanisms.....	264
Hardware-Fixed Arbitration .....	265
Weighted Round Robin Arbitration .....	265
Time-Based, Weighted Round Robin Arbitration (TBWRR).....	266
Loading the Port Arbitration Tables .....	267
Switch Arbitration Example .....	269
<b>Arbitration in Multi-Function Endpoints .....</b>	<b>270</b>
<b>Isochronous Support .....</b>	<b>272</b>
Timing is Everything .....	273
How Timing is Defined.....	274
How Timing is Enforced.....	275

---

# Contents

---

Software Support .....	275
Device Drivers .....	276
Isochronous Broker.....	276
Bringing it all together .....	276
Endpoints .....	276
Switches.....	278
Arbitration Issues .....	278
Timing Issues .....	278
Bandwidth Allocation Problems .....	280
Latency Issues .....	281
Root Complex.....	281
Problem: Snooping .....	281
Snooping Solutions.....	282
Power Management.....	282
Error Handling.....	282

---

## Chapter 8: Transaction Ordering

Introduction.....	285
Definitions.....	286
<b>Simplified Ordering Rules.....</b>	<b>287</b>
Ordering Rules and Traffic Classes (TCs) .....	287
Ordering Rules Based On Packet Type.....	288
The Simplified Ordering Rules Table .....	288
<b>Producer/Consumer Model .....</b>	<b>290</b>
Producer/Consumer Sequence — No Errors .....	291
Producer/Consumer Sequence — Errors.....	295
<b>Relaxed Ordering .....</b>	<b>296</b>
RO Effects on Memory Writes and Messages.....	297
RO Effects on Memory Read Transactions.....	298
<b>Weak Ordering .....</b>	<b>299</b>
Transaction Ordering and Flow Control .....	299
Transaction Stalls .....	300
VC Buffers Offer an Advantage.....	301
<b>ID Based Ordering (IDO) .....</b>	<b>301</b>
The Solution.....	301
When to use IDO.....	302
Software Control.....	303
<b>Deadlock Avoidance.....</b>	<b>303</b>

# Contents

---

## Part Three: Data Link Layer

---

### Chapter 9: DLLP Elements

General .....	307
DLLPs Are Local Traffic .....	308
Receiver handling of DLLPs .....	309
Sending DLLPs .....	309
General.....	309
DLLP Packet Size is Fixed at 8 Bytes.....	310
DLLP Packet Types.....	311
Ack/Nak DLLP Format .....	312
Power Management DLLP Format .....	313
Flow Control DLLP Format.....	314
Vendor-Specific DLLP Format .....	316

---

### Chapter 10: Ack/Nak Protocol

Goal: Reliable TLP Transport.....	317
Elements of the Ack/Nak Protocol.....	320
Transmitter Elements .....	320
NEXT_TRANSMIT_SEQ Counter.....	321
LCRC Generator.....	321
Replay Buffer.....	321
REPLAY_TIMER Count.....	323
REPLAY_NUM Count .....	323
ACKD_SEQ Register .....	323
DLLP CRC Check .....	324
Receiver Elements.....	324
LCRC Error Check .....	325
NEXT_RCV_SEQ Counter.....	326
Sequence Number Check.....	326
NAK_SCHEDULED Flag .....	327
AckNak_LATENCY_TIMER.....	328
Ack/Nak Generator .....	328
Ack/Nak Protocol Details .....	329
Transmitter Protocol Details .....	329
Sequence Number.....	329
32-Bit LCRC .....	329
Replay (Retry) Buffer.....	330
General.....	330
Replay Buffer Sizing.....	330

---

---

# Contents

---

Transmitter's Response to an Ack DLLP .....	331
Ack/Nak Examples .....	331
Example 1.....	331
Example 2.....	332
Transmitter's Response to a Nak.....	333
TLP Replay.....	333
Efficient TLP Replay .....	334
Example of a Nak.....	334
Repeated Replay of TLPs.....	335
General .....	335
Replay Number Rollover.....	336
Replay Timer .....	336
REPLAY_TIMER Equation.....	337
REPLAY_TIMER Summary Table .....	338
Transmitter DLLP Handling .....	340
Receiver Protocol Details .....	340
Physical Layer .....	340
TLP LCRC Check .....	341
Next Received TLP's Sequence Number .....	341
Duplicate TLP .....	342
Out of Sequence TLP.....	342
Receiver Schedules An Ack DLLP .....	342
Receiver Schedules a Nak.....	343
AckNak_LATENCY_TIMER.....	343
AckNak_LATENCY_TIMER Equation .....	344
AckNak_LATENCY_TIMER Summary Table .....	345
More Examples .....	345
Lost TLPs.....	345
Bad Ack .....	347
Bad Nak .....	348
Error Situations Handled by Ack/Nak.....	349
Recommended Priority To Schedule Packets.....	350
Timing Differences for Newer Spec Versions .....	350
Ack Transmission Latency (AckNak Latency) .....	351
2.5 GT/s Operation.....	351
5.0 GT/s Operation.....	352
8.0 GT/s Operation.....	352
Replay Timer .....	353
2.5 GT/s Operation.....	353
5.0 GT/s Operation.....	354
8.0 GT/s Operation.....	354

# Contents

---

Switch Cut-Through Mode .....	354
Background .....	355
A Latency Improvement Option.....	355
Cut-Through Operation .....	356
Example of Cut-Through Operation .....	356

---

## Part Four: Physical Layer

---

### Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

Physical Layer Overview .....	362
Observation.....	364
Transmit Logic Overview .....	364
Receive Logic Overview .....	366
Transmit Logic Details (Gen1 and Gen2 Only) .....	368
Tx Buffer .....	368
Mux and Control Logic .....	368
Byte Striping (for Wide Links) .....	371
Packet Format Rules .....	373
General Rules .....	373
Example: x1 Format.....	374
x4 Format Rules.....	374
Example x4 Format.....	375
Large Link-Width Packet Format Rules .....	376
x8 Packet Format Example .....	376
Scrambler.....	377
Scrambler Algorithm.....	378
Some Scrambler implementation rules:.....	379
Disabling Scrambling .....	379
8b/10b Encoding.....	380
General .....	380
Motivation.....	380
Properties of 10-bit Symbols .....	381
Character Notation .....	382
Disparity.....	383
Definition .....	383
CRD (Current Running Disparity).....	383
Encoding Procedure .....	383
Example Transmission.....	385
Control Characters.....	386
Ordered sets.....	388
General .....	388

# Contents

---

TS1 and TS2 Ordered Set (TS1OS/TS2OS) .....	388
Electrical Idle Ordered Set (EIOS) .....	388
FTS Ordered Set (FTSOS) .....	388
SKP Ordered Set (SOS) .....	389
Electrical Idle Exit Ordered Set (EIEOS) .....	389
Serializer .....	389
Differential Driver .....	389
Transmit Clock (Tx Clock) .....	390
Miscellaneous Transmit Topics .....	390
Logical Idle .....	390
Tx Signal Skew .....	390
Clock Compensation .....	391
Background .....	391
SKIP ordered set Insertion Rules .....	391
Receive Logic Details (Gen1 and Gen2 Only) .....	392
Differential Receiver .....	393
Rx Clock Recovery .....	394
General .....	394
Achieving Bit Lock .....	395
Losing Bit Lock .....	395
Regaining Bit Lock .....	395
Deserializer .....	395
General .....	395
Achieving Symbol Lock .....	396
Receiver Clock Compensation Logic .....	396
Background .....	396
Elastic Buffer's Role .....	397
Lane-to-Lane Skew .....	398
Flight Time Will Vary Between Lanes .....	398
Ordered sets Help De-Skewing .....	398
Receiver Lane-to-Lane De-Skew Capability .....	398
De-Skew Opportunities .....	399
8b/10b Decoder .....	400
General .....	400
Disparity Calculator .....	400
Code Violation and Disparity Error Detection .....	400
General .....	400
Code Violations .....	400
Disparity Errors .....	400
Descrambler .....	402
Some Descrambler Implementation Rules: .....	402
Disabling Descrambling .....	402

# Contents

---

Byte Un-Striping.....	402
Filter and Packet Alignment Check.....	403
Receive Buffer (Rx Buffer) .....	403
<b>Physical Layer Error Handling .....</b>	<b>404</b>
General.....	404
Response of Data Link Layer to Receiver Error .....	404
<b>Active State Power Management .....</b>	<b>405</b>
<b>Link Training and Initialization .....</b>	<b>405</b>

---

## Chapter 12: Physical Layer - Logical (Gen3)

<b>Introduction to Gen3 .....</b>	<b>407</b>
New Encoding Model.....	409
Sophisticated Signal Equalization .....	410
<b>Encoding for 8.0 GT/s .....</b>	<b>410</b>
Lane-Level Encoding .....	410
Block Alignment.....	411
Ordered Set Blocks.....	412
Data Stream and Data Blocks .....	413
Data Block Frame Construction .....	414
Framing Tokens .....	415
Packets .....	415
Transmitter Framing Requirements .....	417
Receiver Framing Requirements .....	419
Recovery from Framing Errors .....	420
<b>Gen3 Physical Layer Transmit Logic.....</b>	<b>421</b>
Multiplexer.....	421
Byte Striping .....	423
Byte Striping x8 Example.....	424
Nullified Packet x8 Example .....	425
Ordered Set Example - SOS.....	426
Transmitter SOS Rules .....	429
Receiver SOS Rules .....	430
Scrambling .....	430
Number of LFSRs.....	430
First Option: Multiple LFSRs .....	431
Second Option: Single LFSR .....	432
Scrambling Rules .....	433
Serializer.....	434
Mux for Sync Header Bits.....	435
<b>Gen3 Physical Layer Receive Logic .....</b>	<b>435</b>
Differential Receiver .....	435
CDR (Clock and Data Recovery) Logic.....	437

---

# Contents

---

Rx Clock Recovery .....	437
Deserializer .....	438
Achieving Block Alignment .....	438
Unaligned Phase .....	439
Aligned Phase .....	439
Locked Phase.....	439
Special Case: Loopback.....	439
Block Type Detection.....	439
Receiver Clock Compensation Logic .....	440
Background.....	440
Elastic Buffer's Role .....	440
Lane-to-Lane Skew .....	442
Flight Time Variance Between Lanes.....	442
De-skew Opportunities.....	442
Receiver Lane-to-Lane De-skew Capability.....	443
Descrambler .....	444
General .....	444
Disabling Descrambling.....	444
Byte Un-Striping.....	445
Packet Filtering.....	446
Receive Buffer (Rx Buffer) .....	446
<b>Notes Regarding Loopback with 128b/130b .....</b>	<b>446</b>

---

## Chapter 13: Physical Layer - Electrical

Backward Compatibility.....	448
Component Interfaces .....	449
Physical Layer Electrical Overview .....	449
High Speed Signaling .....	451
Clock Requirements .....	452
General.....	452
SSC (Spread Spectrum Clocking) .....	453
Refclk Overview .....	455
2.5 GT/s.....	455
5.0 GT/s.....	455
Common Refclk .....	456
Data Clocked Rx Architecture .....	456
Separate Refclks .....	457
8.0 GT/s.....	457
Transmitter (Tx) Specs .....	458
Measuring Tx Signals .....	458
Tx Impedance Requirements.....	459
ESD and Short Circuit Requirements.....	459

# Contents

---

Receiver Detection .....	460
General .....	460
Detecting Receiver Presence.....	460
Transmitter Voltages .....	462
DC Common Mode Voltage.....	462
Full-Swing Differential Voltage.....	462
Differential Notation .....	463
Reduced-Swing Differential Voltage .....	464
Equalized Voltage.....	464
Voltage Margining.....	465
<b>Receiver (Rx) Specs .....</b>	<b>466</b>
Receiver Impedance.....	466
Receiver DC Common Mode Voltage.....	466
Transmission Loss.....	468
AC Coupling.....	468
<b>Signal Compensation .....</b>	<b>468</b>
De-emphasis Associated with Gen1 and Gen2 PCIe .....	468
The Problem.....	468
How Does De-Emphasis Help? .....	469
Solution for 2.5 GT/s.....	470
Solution for 5.0 GT/s.....	472
Solution for 8.0 GT/s - Transmitter Equalization .....	474
Three-Tap Tx Equalizer Required .....	475
Pre-shoot, De-emphasis, and Boost.....	476
Presets and Ratios .....	478
Equalizer Coefficients .....	479
Coefficient Example .....	480
EIEOS Pattern.....	483
Reduced Swing .....	483
Beacon Signaling .....	483
General .....	483
Properties of the Beacon Signal .....	484
<b>Eye Diagram .....</b>	<b>485</b>
Jitter, Noise, and Signal Attenuation .....	485
The Eye Test.....	485
Normal Eye Diagram.....	486
Effects of Jitter.....	487
<b>Transmitter Driver Characteristics .....</b>	<b>489</b>
<b>Receiver Characteristics .....</b>	<b>492</b>
Stressed-Eye Testing.....	492
2.5 and 5.0 GT/s.....	492
8.0 GT/s.....	492

# Contents

---

Receiver (Rx) Equalization .....	493
Continuous-Time Linear Equalization (CTLE) .....	493
Decision Feedback Equalization (DFE) .....	495
<b>Receiver Characteristics</b> .....	<b>497</b>
<b>Link Power Management States</b> .....	<b>500</b>
<hr/>	
<b>Chapter 14: Link Initialization &amp; Training</b>	
<b>Overview</b> .....	<b>506</b>
<b>Ordered Sets in Link Training</b> .....	<b>509</b>
General.....	509
TS1 and TS2 Ordered Sets.....	510
<b>Link Training and Status State Machine (LTSSM)</b> .....	<b>518</b>
General.....	518
Overview of LTSSM States .....	519
Introductions, Examples and State/Substates.....	521
<b>Detect State</b> .....	<b>522</b>
Introduction .....	522
Detailed Detect Substate .....	523
Detect.Quiet .....	523
Detect.Active .....	524
<b>Polling State</b> .....	<b>525</b>
Introduction .....	525
Detailed Polling Substates .....	526
Polling.Active .....	526
Polling.Configuration.....	527
Polling.Compliance .....	529
Compliance Pattern for 8b/10b .....	529
Compliance Pattern for 128b/130b .....	530
Modified Compliance Pattern for 8b/10b.....	532
Modified Compliance Pattern for 128b/130b.....	533
Compliance Pattern .....	537
Modified Compliance Pattern .....	537
<b>Configuration State</b> .....	<b>539</b>
Configuration State — General .....	540
Designing Devices with Links that can be Merged .....	541
Configuration State — Training Examples .....	542
Introduction .....	542
Link Configuration Example 1.....	542
Link Number Negotiation.....	542
Lane Number Negotiation .....	543
Confirming Link and Lane Numbers .....	544

# Contents

---

Link Configuration Example 2.....	545
Link Number Negotiation.....	546
Lane Number Negotiation .....	547
Confirming Link and Lane Numbers .....	548
Link Configuration Example 3: Failed Lane.....	549
Link Number Negotiation.....	549
Lane Number Negotiation .....	550
Confirming Link and Lane Numbers .....	551
Detailed Configuration Substates.....	552
Configuration.Linkwidth.Start .....	553
Downstream Lanes.....	553
Crosslinks.....	554
Upconfiguring the Link Width .....	554
Upstream Lanes .....	556
Crosslinks.....	556
Configuration.Linkwidth.Accept .....	558
Configuration.Lanenum.Wait.....	559
Configuration.Lanenum.Accept .....	560
Configuration.Complete .....	562
Configuration.Idle .....	566
<b>L0 State .....</b>	<b>568</b>
Speed Change .....	568
Link Width Change .....	570
Link Partner Initiated .....	570
<b>Recovery State.....</b>	<b>571</b>
Reasons for Entering Recovery State .....	572
Initiating the Recovery Process.....	572
Detailed Recovery Substates .....	573
Speed Change Example.....	576
Link Equalization Overview .....	577
Phase 0 .....	578
Phase 1 .....	581
Phase 2 .....	583
Phase 3 .....	586
Equalization Notes .....	586
Detailed Equalization Substates .....	587
Recovery.Equalization .....	587
Phase 1 Downstream.....	589
Phase 2 Downstream.....	589
Phase 3 Downstream.....	591
Phase 0 Upstream .....	592
Phase 1 Upstream .....	593

# Contents

---

Phase 2 Upstream .....	593
Phase 3 Upstream .....	594
Recovery.Speed .....	595
Recovery.RcvrCfg .....	598
Recovery.Idle .....	601
L0s State.....	603
L0s Transmitter State Machine .....	603
Tx_L0s.Entry.....	604
Tx_L0s.Idle.....	604
Tx_L0s.FTS.....	604
L0s Receiver State Machine .....	605
Rx_L0s.Entry .....	606
Rx_L0s.Idle .....	606
Rx_L0s.FTS .....	606
L1 State .....	607
L1.Entry .....	608
L1.Idle .....	609
L2 State .....	609
L2.Idle .....	611
L2.TransmitWake.....	612
Hot Reset State.....	612
Disable State.....	613
Loopback State .....	613
Loopback.Entry .....	614
Loopback.Active .....	617
Loopback.Exit .....	618
<b>Dynamic Bandwidth Changes.....</b>	<b>618</b>
Dynamic Link Speed Changes .....	619
Upstream Port Initiates Speed Change .....	622
Speed Change Example.....	622
Software Control of Speed Changes.....	627
Dynamic Link Width Changes.....	629
Link Width Change Example .....	630
<b>Related Configuration Registers.....</b>	<b>638</b>
Link Capabilities Register.....	638
Max Link Speed [3:0].....	639
Maximum Link Width[9:4].....	640
Link Capabilities 2 Register.....	640
Link Status Register .....	641
Current Link Speed[3:0]:.....	641
Negotiated Link Width[9:4] .....	641
Undefined[10].....	642

# Contents

---

Link Training[11] .....	642
Link Control Register .....	642
Link Disable .....	643
Retrain Link .....	643
Extended Synch.....	643

---

## Part Five: Additional System Topics

---

### Chapter 15: Error Detection and Handling

Background .....	648
PCIe Error Definitions .....	650
PCIe Error Reporting .....	650
Baseline Error Reporting.....	650
Advanced Error Reporting (AER) .....	651
Error Classes.....	651
Correctable Errors.....	651
Uncorrectable Errors.....	652
Non-fatal Uncorrectable Errors .....	652
Fatal Uncorrectable Errors.....	652
PCIe Error Checking Mechanisms.....	652
CRC .....	653
Error Checks by Layer.....	655
Physical Layer Errors .....	655
Data Link Layer Errors .....	655
Transaction Layer Errors .....	656
Error Pollution .....	656
Sources of PCI Express Errors.....	657
ECRC Generation and Checking .....	657
TLP Digest.....	659
Variant Bits Not Included in ECRC Mechanism .....	659
Data Poisoning .....	660
Split Transaction Errors .....	662
Unsupported Request (UR) Status .....	663
Completer Abort (CA) Status.....	664
Unexpected Completion .....	664
Completion Timeout .....	665
Link Flow Control Related Errors .....	666
Malformed TLP .....	666
Internal Errors .....	667
The Problem.....	667
The Solution.....	668

# Contents

---

<b>How Errors are Reported .....</b>	<b>668</b>
Introduction .....	668
Error Messages .....	668
Advisory Non-Fatal Errors.....	670
Advisory Non-Fatal Cases.....	671
<b>Baseline Error Detection and Handling.....</b>	<b>674</b>
PCI-Compatible Error Reporting Mechanisms .....	674
General .....	674
Legacy Command and Status Registers .....	675
Baseline Error Handling.....	677
Enabling/Disabling Error Reporting.....	678
Device Control Register .....	680
Device Status Register.....	681
Root's Response to Error Message .....	682
Link Errors .....	683
<b>Advanced Error Reporting (AER) .....</b>	<b>685</b>
Advanced Error Capability and Control .....	686
Handling Sticky Bits .....	688
Advanced Correctable Error Handling .....	688
Advanced Correctable Error Status .....	689
Advanced Correctable Error Masking.....	690
Advanced Uncorrectable Error Handling .....	691
Advanced Uncorrectable Error Status .....	691
Selecting Uncorrectable Error Severity.....	693
Uncorrectable Error Masking.....	694
Header Logging.....	695
Root Complex Error Tracking and Reporting .....	696
Root Complex Error Status Registers .....	696
Advanced Source ID Register .....	697
Root Error Command Register .....	698
<b>Summary of Error Logging and Reporting .....</b>	<b>698</b>
<b>Example Flow of Software Error Investigation .....</b>	<b>699</b>

---

## Chapter 16: Power Management

<b>Introduction.....</b>	<b>704</b>
<b>Power Management Primer.....</b>	<b>705</b>
Basics of PCI PM .....	705
ACPI Spec Defines Overall PM.....	707
System PM States .....	708
Device PM States.....	709
Definition of Device Context.....	709
General .....	709

# Contents

---

PME Context .....	710
Device-Class-Specific PM Specs .....	710
Default Device Class Spec .....	710
Device Class-Specific PM Specs .....	711
Power Management Policy Owner .....	711
PCI Express Power Management vs. ACPI.....	711
PCI Express Bus Driver Accesses PM Registers.....	711
ACPI Driver Controls Non-Standard Embedded Devices .....	712
<b>Function Power Management.....</b>	<b>713</b>
The PM Capability Register Set .....	713
Device PM States.....	713
D0 State—Full On .....	714
Mandatory .....	714
D0 Uninitialized.....	714
D0 Active .....	714
Dynamic Power Allocation (DPA) .....	714
D1 State—Light Sleep.....	716
D2 State—Deep Sleep.....	717
D3—Full Off .....	719
D3Hot State.....	719
D3Cold State.....	721
Function PM State Transitions.....	722
Detailed Description of PCI-PM Registers .....	724
PM Capabilities (PMC) Register.....	724
PM Control and Status Register (PMCSR).....	727
Data Register .....	731
Determining Presence of the Data Register .....	731
Operation of the Data Register .....	731
Multi-Function Devices .....	732
Virtual PCI-to-PCI Bridge Power Data.....	732
<b>Introduction to Link Power Management.....</b>	<b>733</b>
<b>Active State Power Management (ASPM).....</b>	<b>735</b>
Electrical Idle .....	736
Transmitter Entry to Electrical Idle.....	736
Gen1/Gen2 Mode Encoding.....	737
Gen3 Mode Encoding.....	737
Transmitter Exit from Electrical Idle.....	738
Gen1 Mode .....	738
Gen2 Mode .....	738
Gen3 Mode .....	739
Receiver Entry to Electrical Idle.....	740
Detecting Electrical Idle Voltage .....	740

# Contents

---

Inferring Electrical Idle .....	741
Receiver Exit from Electrical Idle .....	742
L0s State.....	744
Entry into L0s .....	745
Entry into L0s .....	745
Flow Control Credits Must be Delivered.....	746
Transmitter Initiates Entry to L0s .....	746
Exit from L0s State .....	746
Transmitter Initiates L0s Exit.....	746
Actions Taken by Switches that Receive L0s Exit.....	746
L1 ASPM State .....	747
Downstream Component Decides to Enter L1 ASPM .....	748
Negotiation Required to Enter L1 ASPM.....	748
Scenario 1: Both Ports Ready to Enter L1 ASPM State .....	748
Downstream Component Requests L1 State .....	748
Upstream Component Response to L1 ASPM Request .....	749
Upstream Component Acknowledges Request to Enter L1.....	749
Downstream Component Sees Acknowledgement.....	749
Upstream Component Receives Electrical Idle .....	749
Scenario 2: Upstream Component Transmits TLP Just Prior to Receiving L1 Request.....	750
TLP Must Be Accepted by Downstream Component .....	751
Upstream Component Receives Request to Enter L1.....	751
Scenario 3: Downstream Component Receives TLP During Negotiation.....	751
Scenario 4: Upstream Component Receives TLP During Negotiation .....	751
Scenario 5: Upstream Component Rejects L1 Request.....	752
Exit from L1 ASPM State .....	753
L1 ASPM Exit Signaling.....	753
Switch Receives L1 Exit from Downstream Component.....	753
Switch Receives L1 Exit from Upstream Component .....	754
ASPM Exit Latency .....	756
Reporting a Valid ASPM Exit Latency .....	756
L0s Exit Latency Update.....	756
L1 Exit Latency Update .....	757
Calculating Latency from Endpoint to Root Complex.....	758
<b>Software Initiated Link Power Management .....</b>	<b>760</b>
D1/D2/D3Hot and the L1 State .....	760
Entering the L1 State .....	760
Exiting the L1 State .....	762
Upstream Component Initiates .....	762
Downstream Component Initiates L1 to L0 Transition .....	763
The L1 Exit Protocol .....	763

# Contents

---

L2/L3 Ready — Removing Power from the Link.....	763
L2/L3 Ready Handshake Sequence.....	764
Exiting the L2/L3 Ready State — Clock and Power Removed.....	767
The L2 State.....	767
The L3 State.....	767
<b>Link Wake Protocol and PME Generation .....</b>	<b>768</b>
The PME Message.....	769
The PME Sequence.....	770
PME Message Back Pressure Deadlock Avoidance .....	770
Background.....	770
The Problem.....	771
The Solution.....	771
The PME Context .....	771
Waking Non-Communicating Links.....	772
Beacon.....	772
WAKE#.....	773
Auxiliary Power .....	775
<b>Improving PM Efficiency .....</b>	<b>776</b>
Background .....	776
OBFF (Optimized Buffer Flush and Fill) .....	776
The Problem.....	776
The Solution.....	778
Using the WAKE# Pin.....	779
Using the OBFF Message.....	780
LTR (Latency Tolerance Reporting) .....	784
LTR Registers.....	784
LTR Messages.....	786
Guidelines Regarding LTR Use .....	786
LTR Example .....	789

---

## Chapter 17: Interrupt Support

<b>Interrupt Support Background .....</b>	<b>794</b>
General.....	794
Two Methods of Interrupt Delivery .....	794
<b>The Legacy Model.....</b>	<b>796</b>
General.....	796
Changes to Support Multiple Processors .....	798
Legacy PCI Interrupt Delivery .....	800
Device INTx# Pins .....	800
Determining INTx# Pin Support .....	801
Interrupt Routing.....	802
Associating the INTx# Line to an IRQ Number .....	802

---

# Contents

---

INTx# Signaling .....	803
Interrupt Disable.....	803
Interrupt Status .....	804
Virtual INTx Signaling .....	805
General .....	805
Virtual INTx Wire Delivery.....	806
INTx Message Format.....	807
Mapping and Collapsing INTx Messages .....	808
INTx Mapping.....	808
INTx Collapsing.....	810
INTx Delivery Rules.....	812
<b>The MSI Model.....</b>	<b>812</b>
The MSI Capability Structure.....	812
Capability ID .....	814
Next Capability Pointer .....	814
Message Control Register.....	814
Message Address Register.....	816
Message Data Register .....	817
Mask Bits Register and Pending Bits Register.....	817
Basics of MSI Configuration.....	817
Basics of Generating an MSI Interrupt Request .....	820
Multiple Messages .....	820
<b>The MSI-X Model.....</b>	<b>821</b>
General.....	821
MSI-X Capability Structure .....	822
MSI-X Table.....	824
Pending Bit Array .....	825
<b>Memory Synchronization When Interrupt Handler Entered .....</b>	<b>826</b>
The Problem.....	826
One Solution .....	827
An MSI Solution.....	827
Traffic Classes Must Match .....	828
<b>Interrupt Latency.....</b>	<b>829</b>
<b>MSI May Result In Errors.....</b>	<b>829</b>
<b>Some MSI Rules and Recommendations .....</b>	<b>830</b>
<b>Special Consideration for Base System Peripherals .....</b>	<b>830</b>
Example Legacy System.....	831
<hr/>	
<b>Chapter 18: System Reset</b>	
<b>Two Categories of System Reset.....</b>	<b>833</b>
<b>Conventional Reset.....</b>	<b>834</b>
Fundamental Reset .....	834

# Contents

---

PERST# Fundamental Reset Generation .....	835
Autonomous Reset Generation.....	835
Link Wakeup from L2 Low Power State .....	836
Hot Reset (In-band Reset).....	837
Response to Receiving Hot Reset .....	837
Switches Generate Hot Reset on Downstream Ports.....	838
Bridges Forward Hot Reset to the Secondary Bus .....	838
Software Generation of Hot Reset.....	838
Software Can Disable the Link .....	840
<b>Function Level Reset (FLR) .....</b>	<b>842</b>
Time Allowed .....	844
Behavior During FLR .....	845
<b>Reset Exit.....</b>	<b>846</b>

---

## Chapter 19: Hot Plug and Power Budgeting

<b>Background .....</b>	<b>848</b>
<b>Hot Plug in the PCI Express Environment.....</b>	<b>848</b>
Surprise Removal Notification.....	849
Differences between PCI and PCIe Hot Plug.....	849
<b>Elements Required to Support Hot Plug.....</b>	<b>852</b>
Software Elements .....	852
Hardware Elements .....	853
<b>Card Removal and Insertion Procedures.....</b>	<b>855</b>
On and Off States .....	855
Turning Slot Off .....	855
Turning Slot On.....	855
Card Removal Procedure.....	856
Card Insertion Procedure.....	857
<b>Standardized Usage Model .....</b>	<b>858</b>
Background .....	858
Standard User Interface .....	859
Attention Indicator .....	859
Power Indicator.....	860
Manually Operated Retention Latch and Sensor .....	861
Electromechanical Interlock (optional).....	862
Software User Interface.....	862
Attention Button .....	862
Slot Numbering Identification .....	862
<b>Standard Hot Plug Controller Signaling Interface.....</b>	<b>863</b>
<b>The Hot-Plug Controller Programming Interface.....</b>	<b>864</b>
Slot Capabilities.....	865
Slot Power Limit Control .....	867

# Contents

---

Slot Control .....	868
Slot Status and Events Management.....	870
Add-in Card Capabilities.....	872
Quiescing Card and Driver .....	873
General.....	873
Pausing a Driver (Optional) .....	874
Quiescing a Driver That Controls Multiple Devices .....	874
Quiescing a Failed Card.....	874
The Primitives.....	874
Introduction to Power Budgeting .....	876
The Power Budgeting Elements .....	877
System Firmware .....	877
The Power Budget Manager.....	878
Expansion Ports.....	878
Add-in Devices.....	879
Slot Power Limit Control.....	881
Expansion Port Delivers Slot Power Limit.....	881
Expansion Device Limits Power Consumption.....	883
The Power Budget Capabilities Register Set.....	883

---

## Chapter 20: Updates for Spec Revision 2.1

Changes for PCIe Spec Rev 2.1.....	887
System Redundancy Improvement: Multi-casting.....	888
Multicast Capability Registers .....	889
Multicast Capability .....	889
Multicast Control .....	890
Multicast Base Address.....	891
MC Receive .....	892
MC Block All .....	892
MC Block Untranslated.....	892
Multicast Example .....	893
MC Overlay BAR .....	894
Overlay Example.....	895
Routing Multicast TLPs.....	896
Congestion Avoidance .....	897
Performance Improvements .....	897
AtomicOps .....	897
TPH (TLP Processing Hints).....	899
TPH Examples .....	900
Device Write to Host Read .....	900
Host Write to Device Read .....	902
Device to Device .....	903

# Contents

---

TPH Header Bits .....	904
Steering Tags .....	906
TLP Prefixes.....	908
IDO (ID-based Ordering).....	909
ARI (Alternative Routing-ID Interpretation).....	909
<b>Power Management Improvements .....</b>	<b>910</b>
DPA (Dynamic Power Allocation.....	910
LTR (Latency Tolerance Reporting) .....	910
OBFF (Optimized Buffer Flush and Fill) .....	910
ASPM Options.....	910
<b>Configuration Improvements .....</b>	<b>911</b>
Internal Error Reporting .....	911
Resizable BARs.....	911
Capability Register .....	912
Control Register .....	912
Simplified Ordering Table .....	914

---

## Appendices

---

<b>Appendix A: Debugging PCIe Traffic with LeCroy Tools</b>	
Overview.....	917
<b>Pre-silicon Debugging .....</b>	<b>918</b>
RTL Simulation Perspective .....	918
PCI Express RTL Bus Monitor .....	918
RTL vector export to PETracer Application.....	918
<b>Post-Silicon Debug .....</b>	<b>919</b>
Oscilloscope .....	919
Protocol Analyzer .....	920
Logic Analyzer .....	921
<b>Using a Protocol Analyzer Probing Option .....</b>	<b>921</b>
<b>Viewing Traffic Using the PETracer Application.....</b>	<b>924</b>
CATC Trace Viewer.....	924
LTSSM Graphs.....	927
Flow Control Credit Tracking .....	928
Bit Tracer .....	929
Analysis overview .....	931
<b>Traffic generation.....</b>	<b>931</b>
Pre-Silicon .....	931
Post-Silicon.....	931
Exerciser Card .....	931
PTC card .....	932

# Contents

---

Conclusion.....	933
-----------------	-----

## Appendix B: Markets & Applications for PCI Express

Introduction.....	935
PCI Express IO Virtualization Solutions.....	937
Multi-Root (MR) PCIe Switch Solution .....	938
PCIe Beyond Chip-to-Chip Interconnect .....	939
SSD/Storage IO Expansion Boxes.....	940
PCIe in SSD Modules for Servers.....	940
Conclusion.....	942

---

## Appendix C: Implementing Intelligent Adapters and Multi-Host Systems With PCI Express Technology

Introduction.....	943
Usage Models .....	944
Intelligent Adapters .....	944
Host Failover .....	944
Multiprocessor Systems .....	945
The History Multi-Processor Implementations Using PCI .....	945
Implementing Multi-host/Intelligent Adapters in PCI Express Base Systems.....	947
Example: Implementing Intelligent Adapters in a PCI Express Base System .....	950
Example: Implementing Host Failover in a PCI Express System .....	952
Example: Implementing Dual Host in a PCI Express Base System.....	955
Summary .....	957
Address Translation .....	958
Direct Address Translation.....	959
Lookup Table Based Address Translation .....	959
Downstream BAR Limit Registers.....	960
Forwarding 64bit Address Memory Transactions .....	961

---

## Appendix D: Locked Transactions

Introduction.....	963
Background .....	963
The PCI Express Lock Protocol.....	964
Lock Messages — The Virtual Lock Signal .....	964
The Lock Protocol Sequence — an Example .....	965
The Memory Read Lock Operation.....	965
Read Data Modified and Written to Target and Lock Completes.....	967
Notification of an Unsuccessful Lock .....	970

# **Contents**

---

<b>Summary of Locking Rules.....</b>	<b>970</b>
Rules Related To the Initiation and Propagation of Locked Transactions .....	970
Rules Related to Switches .....	971
Rules Related To PCI Express/PCI Bridges.....	972
Rules Related To the Root Complex.....	972
Rules Related To Legacy Endpoints.....	972
Rules Related To PCI Express Endpoints .....	972
<b>Glossary .....</b>	<b>973</b>

# Figures

---

1-1	Legacy PCI Bus-Based Platform .....	12
1-2	PCI Bus Arbitration .....	13
1-3	Simple PCI Bus Transfer .....	15
1-4	PCI Reflected-Wave Signaling .....	17
1-5	33 MHz PCI System, Including a PCI-to-PCI Bridge .....	18
1-6	PCI Transaction Models.....	19
1-7	PCI Transaction Retry Mechanism.....	21
1-8	PCI Transaction Disconnect Mechanism.....	23
1-9	PCI Error Handling .....	24
1-10	Address Space Mapping .....	26
1-11	Configuration Address Register .....	27
1-12	PCI Configuration Header Type 1 (Bridge) .....	28
1-13	PCI Configuration Header Type 0 (not a Bridge) .....	29
1-14	66 MHz PCI Bus Based Platform .....	30
1-15	66 MHz/133 MHz PCI-X Bus Based Platform .....	32
1-16	Example PCI-X Burst Memory Read Bus Cycle .....	33
1-17	PCI-X Split Transaction Protocol.....	34
1-18	Inherent Problems in a Parallel Design .....	36
1-19	Source-Synchronous Clocking Model .....	38
2-1	Dual-Simplex Link.....	40
2-2	One Lane .....	40
2-3	Parallel Bus Limitations .....	42
2-4	Differential Signaling .....	44
2-5	Simple PLL Block Diagram .....	45
2-6	Example PCIe Topology .....	47
2-7	Configuration Headers .....	50
2-8	Topology Example.....	51
2-9	Example Results of System Enumeration .....	52
2-10	Low-Cost PCIe System.....	53
2-11	Server PCIe System.....	54
2-12	PCI Express Device Layers .....	56
2-13	Switch Port Layers .....	57
2-14	Detailed Block Diagram of PCI Express Device's Layers .....	58
2-15	TLP Origin and Destination .....	62
2-16	TLP Assembly .....	63
2-17	TLP Disassembly.....	64
2-18	Non-Posted Read Example.....	65
2-19	Non-Posted Locked Read Transaction Protocol.....	67
2-20	Non-Posted Write Transaction Protocol.....	68
2-21	Posted Memory Write Transaction Protocol.....	69
2-22	QoS Example .....	71
2-23	Flow Control Basics .....	72

# Figures

---

2-24	DLLP Origin and Destination .....	73
2-25	Data Link Layer Replay Mechanism.....	74
2-26	TLP and DLLP Structure at the Data Link Layer.....	75
2-27	Non-Posted Transaction with Ack/Nak Protocol .....	76
2-28	TLP and DLLP Structure at the Physical Layer.....	77
2-29	Physical Layer Electrical .....	79
2-30	Ordered Sets Origin and Destination .....	80
2-31	Ordered-Set Structure .....	80
2-32	Memory Read Request Phase.....	81
2-33	Completion with Data Phase .....	83
3-1	Example System .....	87
3-2	PCI Compatible Configuration Register Space .....	89
3-3	4KB Configuration Space per PCI Express Function.....	90
3-4	Configuration Address Port at 0CF8h .....	92
3-5	Single-Root System.....	95
3-6	Multi-Root System .....	97
3-7	Type 0 Configuration Read and Write Request Headers .....	100
3-8	Type 1 Configuration Read and Write Request Headers .....	101
3-9	Example Configuration Read Access.....	104
3-10	Topology View At Startup .....	105
3-11	Root Control Register in PCIe Capability Block.....	108
3-12	Header Type Register.....	108
3-13	Single-Root System .....	113
3-14	Multi-Root System .....	116
3-15	Partial Screenshot of MindShare Arbor.....	118
4-1	Generic Memory And IO Address Maps .....	125
4-2	BARs in Configuration Space.....	127
4-3	PCI Express Devices And Type 0 And Type 1 Header Use .....	128
4-4	32-Bit Non-Prefetchable Memory BAR Set Up.....	130
4-5	64-Bit Prefetchable Memory BAR Set Up .....	132
4-6	IO BAR Set Up.....	134
4-7	Example Topology for Setting Up Base and Limit Values .....	137
4-8	Example Prefetchable Memory Base/Limit Register Values .....	138
4-9	Example Non-Prefetchable Memory Base/Limit Register Values .....	140
4-10	Example IO Base/Limit Register Values.....	142
4-11	Final Example Address Routing Setup.....	145
4-12	Multi-Port PCIe Devices Have Routing Responsibilities.....	146
4-13	PCI Express Transaction Request And Completion TLPs.....	149
4-14	Transaction Layer Packet Generic 3DW And 4DW Headers .....	152
4-15	3DW TLP Header - ID Routing Fields .....	156
4-16	4DW TLP Header - ID Routing Fields .....	156
4-17	Switch Checks Routing Of An Inbound TLP Using ID Routing .....	158

# Figures

---

4-18	3DW TLP Header - Address Routing Fields.....	159
4-19	4DW TLP Header - Address Routing Fields.....	160
4-20	Endpoint Checks Incoming TLP Address.....	161
4-21	Switch Checks Routing Of An Inbound TLP Using Address .....	162
4-22	4DW Message TLP Header - Implicit Routing Fields .....	164
5-1	TLP And DLLP Packets .....	170
5-2	PCIe TLP Assembly/Disassembly .....	173
5-3	Generic TLP Header Fields .....	175
5-4	Using First DW and Last DW Byte Enable Fields.....	182
5-5	Transaction Descriptor Fields .....	183
5-6	System IO Map .....	185
5-7	3DW IO Request Header Format.....	185
5-8	3DW And 4DW Memory Request Header Formats .....	188
5-9	3DW Configuration Request And Header Format .....	193
5-10	3DW Completion Header Format .....	197
5-11	4DW Message Request Header Format.....	203
5-12	Vendor-Defined Message Header .....	211
5-13	LTR Message Header .....	212
5-14	OBFF Message Header.....	213
6-1	Location of Flow Control Logic .....	217
6-2	Flow Control Buffer Organization .....	218
6-3	Physical Layer Reports That It's Ready .....	222
6-4	The Data Link Control & Management State Machine .....	223
6-5	INIT1 Flow Control DLLP Format and Contents .....	224
6-6	Devices Send InitFC1 in the DL_Init State .....	225
6-7	FC Values Registered - Send InitFC2s, Report DL_Up .....	226
6-8	Flow Control Elements .....	228
6-9	Types and Format of Flow Control DLLPs.....	229
6-10	Flow Control Elements Following Initialization.....	231
6-11	Flow Control Elements After First TLP Sent .....	232
6-12	Flow Control Elements with Flow Control Buffer Filled.....	234
6-13	Flow Control Rollover Problem.....	235
6-14	Buffer Overflow Error Check .....	236
6-15	Flow Control Update Example .....	238
6-16	Update Flow Control Packet Format and Contents .....	239
7-1	Virtual Channel Capability Registers .....	246
7-2	Traffic Class Field in TLP Header .....	247
7-3	TC to VC Mapping Example .....	249
7-4	Multiple VCs Supported by a Device .....	250
7-5	Extended VCs Supported Field .....	251
7-6	VC Arbitration Example .....	253
7-7	Strict Priority Arbitration.....	254

# **Figures**

---

7-8	Low-Priority Extended VCs .....	255
7-9	VC Arbitration Capabilities.....	256
7-10	VC Arbitration Priorities .....	257
7-11	WRR VC Arbitration Table.....	258
7-12	VC Arbitration Table Offset and Load VC Arbitration Table Fields .....	259
7-13	Loading the VC Arbitration Table Entries .....	260
7-14	Port Arbitration Concept .....	262
7-15	Port Arbitration Tables for Each VC .....	263
7-16	Port Arbitration Buffering .....	264
7-17	Software Selects Port Arbitration Scheme.....	265
7-18	Maximum Time Slots Register.....	267
7-19	Format of Port Arbitration Tables .....	268
7-20	Arbitration Examples in a Switch.....	270
7-21	Simple Multi-Function Arbitration .....	271
7-22	QoS Support in Multi-Function Arbitration .....	272
7-23	Example Application of Isochronous Transaction.....	274
7-24	Example Isochronous System .....	277
7-25	Injection of Isochronous Packets .....	279
7-26	Over-Subscribing the Bandwidth .....	280
7-27	Bandwidth Congestion .....	281
8-1	Example Producer/Consumer Topology.....	291
8-2	Producer/Consumer Sequence Example — Part 1.....	293
8-3	Producer/Consumer Sequence Example — Part 2.....	294
8-4	Producer/Consumer Sequence with Error .....	296
8-5	Relaxed Ordering Bit in a 32-bit Header .....	297
8-6	Strongly Ordered Example Results in Temporary Stall.....	300
8-7	Different Sources are Unlikely to Have Dependencies .....	302
8-8	IDO Attribute in 64-bit Header.....	303
9-1	Data Link Layer Sends A DLLP.....	308
9-2	Generic Data Link Layer Packet Format .....	310
9-3	Ack Or Nak DLLP Format.....	312
9-4	Power Management DLLP Format .....	314
9-5	Flow Control DLLP Format.....	315
9-6	Vendor-Specific DLLP Format.....	316
10-1	Data Link Layer.....	318
10-2	Overview of the Ack/Nak Protocol.....	319
10-3	Elements of the Ack/Nak Protocol .....	320
10-4	Transmitter Elements Associated with the Ack/Nak Protocol .....	322
10-5	Receiver Elements Associated with the Ack/Nak Protocol .....	325
10-6	Examples of Sequence Number Ranges .....	327
10-7	Ack Or Nak DLLP Format.....	328
10-8	Example 1 - Example of Ack .....	332

10-9	Example 2 - Ack with Sequence Number Rollover .....	333
10-10	Example of a Nak.....	335
10-11	Gen1 Unadjusted REPLAY_TIMER Values.....	339
10-12	Ack/Nak Receiver Elements.....	341
10-13	Handling Lost TLPs.....	346
10-14	Handling Bad Ack .....	347
10-15	Handling Bad Nak.....	349
10-16	Switch Cut-Through Mode Showing Error Handling.....	357
11-1	PCIe Port Layers .....	362
11-2	Logical and Electrical Sub-Blocks of the Physical Layer.....	363
11-3	Physical Layer Transmit Details.....	365
11-4	Physical Layer Receive Logic Details.....	367
11-5	Physical Layer Transmit Logic Details (Gen1 and Gen2 Only) .....	369
11-6	Transmit Logic Multiplexer.....	370
11-7	TLP and DLLP Packet Framing with Start and End Control Characters .....	371
11-8	x1 Byte Striping .....	372
11-9	x4 Byte Striping .....	372
11-10	x8 Byte Striping with DWord Parallel Data .....	373
11-11	x1 Packet Format.....	374
11-12	x4 Packet Format.....	375
11-13	x8 Packet Format.....	377
11-14	Scrambler .....	378
11-15	Example of 8-bit Character 00h Encoding.....	381
11-16	8b/10b Nomenclature.....	382
11-17	8-bit to 10-bit (8b/10b) Encoder.....	384
11-18	Example 8b/10b Encodings .....	385
11-19	Example 8b/10b Transmission .....	386
11-20	SKIP Ordered Set .....	392
11-21	Physical Layer Receive Logic Details (Gen1 and Gen2 Only).....	393
11-22	Receiver Logic's Front End Per Lane .....	394
11-23	Receiver's Link De-Skew Logic .....	399
11-24	8b/10b Decoder per Lane .....	401
11-25	Example of Delayed Disparity Error Detection .....	401
11-26	Example of x8 Byte Un-Striping .....	403
12-1	8b/10b Lane Encoding.....	409
12-2	128b/130b Block Encoding.....	410
12-3	Sync Header Data Block Example.....	411
12-4	Gen3 Mode EIEOS Symbol Pattern.....	411
12-5	Gen3 x1 Ordered Set Block Example .....	412
12-6	Gen3 FTS Ordered Set Example .....	413
12-7	Gen3 x1 Frame Construction Example .....	414
12-8	Gen3 Frame Token Examples .....	417

# Figures

---

12-9	AER Correctable Error Register.....	421
12-10	Gen3 Physical Layer Transmitter Details.....	422
12-11	Gen3 Byte Striping x4.....	424
12-12	Gen3 x8 Example: TLP Straddles Block Boundary .....	425
12-13	Gen3 x8 Nullified Packet .....	426
12-14	Gen3 x1 Ordered Set Construction .....	427
12-15	Gen3 x8 Skip Ordered Set (SOS) Example .....	428
12-16	Gen3 Per-Lane LFSR Scrambling Logic.....	431
12-17	Gen3 Single-LFSR Scrambler .....	433
12-18	Gen3 Physical Layer Receiver Details.....	436
12-19	Gen3 CDR Logic.....	437
12-20	EIEOS Symbol Pattern.....	438
12-21	Gen3 Elastic Buffer Logic.....	441
12-22	Receiver Link De-Skew Logic .....	444
12-23	Physical Layer Receive Logic Details.....	445
13-1	Electrical Sub-Block of the Physical Layer .....	450
13-2	Differential Transmitter/Receiver.....	451
13-3	Differential Common-Mode Noise Rejection .....	452
13-4	SSC Motivation.....	454
13-5	Signal Rate Less Than Half the Clock Rate .....	454
13-6	SSC Modulation Example.....	455
13-7	Shared Refclk Architecture.....	456
13-8	Data Clocked Rx Architecture .....	457
13-9	Separate Refclk Architecture.....	457
13-10	Test Circuit Measurement Channels.....	458
13-11	Receiver Detection Mechanism.....	461
13-12	Differential Signaling .....	463
13-13	Differential Peak-to-Peak (VDIFFp-p) and Peak (VDIFFp) Voltages.....	464
13-14	Transmit Margin Field in Link Control 2 Register.....	465
13-15	Receiver DC Common-Mode Voltage Adjustment .....	467
13-16	Transmission with De-emphasis .....	469
13-17	Benefit of De-emphasis at the Receiver .....	471
13-18	Benefit of De-emphasis at Receiver Shown With Differential Signals.....	472
13-19	De-emphasis Options for 5.0 GT/s .....	473
13-20	Reduced-Swing Option for 5.0 GT/s with No De-emphasis .....	474
13-21	3-Tap Tx Equalizer.....	475
13-22	Tx 3-Tap Equalizer Shaping of an Output Pulse.....	476
13-23	8.0 GT/s Tx Voltage Levels .....	477
13-24	Tx 3-Tap Equalizer Output.....	482
13-25	Example Beacon Signal .....	484
13-26	Transmitter Eye Diagram .....	486
13-27	Rx Normal Eye (No De-emphasis) .....	488

# Figures

---

13-28	Rx Bad Eye (No De-emphasis).....	488
13-29	Rx Discrete-Time Linear Equalizer (DLE).....	494
13-30	Rx Continuous-Time Linear Equalizer (CTLE) .....	494
13-31	Effect of Rx Continuous-Time Linear Equalizer (CTLE) on Received Signal..	495
13-32	Rx 1-Tap DFE.....	495
13-33	Rx 2-Tap DFE.....	497
13-34	2.5 GT/s Receiver Eye Diagram .....	499
13-35	L0 Full-On Link State .....	500
13-36	L0s Low Power Link State .....	501
13-37	L1 Low Power Link State.....	502
13-38	L2 Low Power Link State.....	503
13-39	L3 Link Off State .....	504
14-1	Link Training and Status State Machine Location .....	506
14-2	Lane Reversal Example (Support Optional) .....	508
14-3	Polarity Inversion Example (Support Required).....	509
14-4	TS1 and TS2 Ordered Sets When In Gen1 or Gen2 Mode .....	510
14-5	TS1 and TS2 Ordered Set Block When In Gen3 Mode of Operation .....	511
14-6	Link Training and Status State Machine (LTSSM).....	519
14-7	States Involved in Initial Link Training at 2.5 Gb/s.....	522
14-8	Detect State Machine .....	523
14-9	Polling State Machine.....	525
14-10	Polling State Machine with Legacy Speed Change.....	528
14-11	Link Control 2 Register .....	536
14-12	Link Control 2 Register's "Enter Compliance" Bit .....	539
14-13	Link and Lane Number Encoding in TS1/TS2.....	540
14-14	Combining Lanes to Form Wider Links (Link Merging).....	541
14-15	Example 1 - Steps 1 and 2 .....	543
14-16	Example 1 - Steps 3 and 4 .....	544
14-17	Example 1 - Steps 5 and 6 .....	545
14-18	Example 2 - Step 1.....	546
14-19	Example 2 - Step 2.....	547
14-20	Example 2 - Steps 3, 4 and 5 .....	548
14-21	Example 3 - Steps 1 and 2 .....	550
14-22	Example 3 - Steps 3 and 4 .....	551
14-23	Example 3 - Steps 5 and 6 .....	552
14-24	Configuration State Machine .....	553
14-25	Link Control Register .....	569
14-26	Link Control 2 Register .....	569
14-27	Recovery State Machine.....	573
14-28	EC Field in TS1s and TS2s for 8.0 GT/s.....	578
14-29	Equalization Control Registers .....	579
14-30	Equalization Process: Starting Point .....	581

# **Figures**

---

14-31	Equalization Process: Initiating Phase 2 .....	583
14-32	Equalization Coefficients Exchanged .....	584
14-33	3-Tap Transmitter Equalization.....	585
14-34	Equalization Process: Adjustments During Phase 2.....	585
14-35	Equalization Process: Adjustments During Phase 3.....	586
14-36	Link Status 2 Register.....	588
14-37	Link Control 3 Register .....	588
14-38	TS1s - Rejecting Coefficient Values .....	590
14-39	Link Status Register.....	597
14-40	L0s Tx State Machine.....	603
14-41	L0s Receiver State Machine .....	605
14-42	L1 State Machine .....	608
14-43	L2 State Machine .....	611
14-44	Loopback State Machine .....	614
14-45	LTSSM Overview .....	620
14-46	TS1 Contents.....	621
14-47	TS2 Contents.....	621
14-48	Recovery Sub-States .....	622
14-49	Speed Change - Initiated.....	623
14-50	Speed Change - Part 2 .....	624
14-51	Speed Change - Part 3 .....	625
14-52	Bandwidth Change Status Bits .....	625
14-53	Bandwidth Notification Capability.....	626
14-54	Bandwidth Change Notification Bits .....	626
14-55	Speed Change Finish.....	627
14-56	Link Control 2 Register .....	628
14-57	Link Control Register .....	629
14-58	TS2 Contents.....	630
14-59	Link Width Change Example.....	631
14-60	Link Width Change LTSSM Sequence.....	631
14-61	Simplified Configuration Substates .....	632
14-62	Link Width Change - Start.....	633
14-63	Link Width Change - Recovery.Idle.....	634
14-64	Marking Active Lanes .....	635
14-65	Response to Lane Number Changes .....	636
14-66	Link Width Change - Finish .....	637
14-67	Link Control Register .....	638
14-68	Link Capabilities Register.....	639
14-69	Link Capabilities 2 Register.....	640
14-70	Link Status Register .....	642
14-71	Link Control Register .....	644
15-1	PCI Error Handling .....	649

# Figures

---

15-2	Scope of PCI Express Error Checking and Reporting .....	653
15-3	ECRC Usage Example .....	654
15-4	Location of Error-Related Configuration Registers .....	658
15-5	TLP Digest Bit in a Completion Header .....	659
15-6	The Error/Poisoned Bit in a Completion Header .....	660
15-7	Completion Status Field within the Completion Header .....	662
15-8	Device Control Register 2 .....	665
15-9	Error Message Format.....	669
15-10	Device Capabilities Register.....	670
15-11	Role-Based Error Reporting Example .....	672
15-12	Advanced Source ID Register .....	672
15-13	Command Register in Configuration Header .....	675
15-14	Status Register in Configuration Header .....	676
15-15	PCI Express Capability Structure .....	678
15-16	Device Control Register Fields Related to Error Handling .....	681
15-17	Device Status Register Bit Fields Related to Error Handling .....	682
15-18	Root Control Register .....	683
15-19	Link Control Register - Force Link Retraining .....	684
15-20	Link Training Status in the Link Status Register.....	685
15-21	Advanced Error Capability Structure.....	686
15-22	The Advanced Error Capability and Control Register.....	687
15-23	Advanced Correctable Error Status Register .....	689
15-24	Advanced Correctable Error Mask Register .....	690
15-25	Advanced Uncorrectable Error Status Register .....	691
15-26	Advanced Uncorrectable Error Severity Register .....	694
15-27	Advanced Uncorrectable Error Mask Register .....	694
15-28	Root Error Status Register .....	697
15-29	Advanced Source ID Register .....	698
15-30	Advanced Root Error Command Register .....	698
15-31	Flow Chart of Error Handling Within a Function .....	699
15-32	Error Investigation Example System .....	701
16-1	Relationship of OS, Device Drivers, Bus Driver, PCI Express Registers, and ACPI712 .....	
16-2	PCI Power Management Capability Register Set.....	713
16-3	Dynamic Power Allocation Registers .....	715
16-4	DPA Capability Register .....	716
16-5	DPA Status Register .....	716
16-6	PCIe Function D-State Transitions .....	722
16-7	PCI Function's PM Registers.....	724
16-8	PM Registers .....	732
16-9	Gen1/Gen2 Mode EIOS Pattern .....	737
16-10	Gen3 Mode EIOS Pattern.....	737

# **Figures**

---

16-11	Gen1/Gen2 Mode EIEOS Symbol Pattern .....	739
16-12	128b/130b EIEOS Block .....	740
16-13	ASPM Link State Transitions .....	742
16-14	ASPM Support .....	743
16-15	Active State PM Control Field .....	744
16-16	Only Upstream Ports Initiate L1 ASPM .....	747
16-17	Negotiation Sequence Required to Enter L1 Active State PM .....	750
16-18	Negotiation Sequence Resulting in Rejection to Enter L1 ASPM State .....	752
16-19	Switch Behavior When Downstream Component Signals L1 Exit.....	754
16-20	Switch Behavior When Upstream Component Signals L1 Exit .....	755
16-21	Config. Registers for ASPM Exit Latency Management and Reporting.....	757
16-22	Example of Total L1 Latency .....	759
16-23	Devices Transition to L1 When Software Changes their Power Level from D0760	
16-24	Procedure Used to Transition a Link from the L0 to L1 State .....	762
16-25	Link States Transitions Associated with Preparing Devices for Removal of the Reference Clock and Power764	
16-26	Negotiation for Entering L2/L3 Ready State.....	766
16-27	State Transitions from L2/L3 Ready When Power is Removed .....	767
16-28	PME Message Format.....	769
16-29	WAKE# Signal Implementations.....	774
16-30	Auxiliary Current Enable for Devices Not Supporting PMEs .....	775
16-31	Poor System Idle Time .....	777
16-32	Improved System Idle Time .....	777
16-33	OBFF Signaling Example .....	778
16-34	WAKE# Pin OBFF Signaling .....	779
16-35	OBFF Message Contents .....	781
16-36	OBFF Support Indication .....	782
16-37	OBFF Enable Register .....	783
16-38	LTR Capability Status .....	785
16-39	LTR Enable.....	785
16-40	LTR Message Format.....	788
16-41	LTR Example .....	789
16-42	LTR - Change but no Update .....	790
16-43	LTR - Change with Update .....	791
16-44	LTR - Link Down Case .....	791
17-1	PCI Interrupt Delivery .....	795
17-2	Interrupt Delivery Options in PCIe System .....	796
17-3	Legacy Interrupt Example .....	797
17-4	APIC Model for Interrupt Delivery .....	799
17-5	Interrupt Registers in PCI Configuration Header.....	801
17-6	INTx Signal Routing is Platform Specific.....	803

# Figures

---

17-7	Configuration Command Register — Interrupt Disable Field.....	804
17-8	Configuration Status Register — Interrupt Status Field .....	805
17-9	Example of INTx Messages to Virtualize INTA#-INTD# Signal Transitions	806
17-10	INTx Message Format and Type .....	807
17-11	Example of INTx Mapping .....	810
17-12	Switch Uses Bridge Mapping of INTx Messages .....	811
17-13	MSI Capability Structure Variations.....	813
17-14	Message Control Register .....	814
17-15	Device MSI Configuration Process.....	819
17-16	Format of Memory Write Transaction for Native-Device MSI Delivery .....	821
17-17	MSI-X Capability Structure .....	822
17-18	Location of MSI-X Table .....	824
17-19	MSI-X Table Entries .....	825
17-20	Pending Bit Array .....	826
17-21	Memory Synchronization Problem .....	827
17-22	MSI Delivery.....	829
17-23	PCI Express System with PCI-Based IO Controller Hub .....	831
18-1	PERST# Generation .....	836
18-2	TS1 Ordered-Set Showing the Hot Reset Bit.....	837
18-3	Switch Generates Hot Reset on One Downstream Port .....	838
18-4	Switch Generates Hot Reset on All Downstream Ports .....	839
18-5	Secondary Bus Reset Register to Generate Hot Reset .....	840
18-6	Link Control Register .....	841
18-7	TS1 Ordered-Set Showing Disable Link Bit .....	842
18-8	Function-Level Reset Capability.....	843
18-9	Function-Level Reset Initiate Bit.....	843
19-1	PCI Hot Plug Elements .....	850
19-2	PCI Express Hot-Plug Elements .....	851
19-3	Hot Plug Control Functions within a Switch.....	864
19-4	PCIe Capability Registers Used for Hot-Plug .....	865
19-5	Slot Capabilities Register .....	866
19-6	Slot Control Register .....	868
19-7	Slot Status Register .....	870
19-8	Device Capabilities Register.....	873
19-9	Power Budget Registers.....	878
19-10	Elements Involved in Power Budget .....	880
19-11	Slot Power Limit Sequence .....	882
19-12	Power Budget Capability Registers .....	884
19-13	Power Budget Data Field Format and Definition .....	885
20-1	Multicast System Example .....	888
20-2	Multicast Capability Registers .....	889

# **Figures**

---

20-3	Multicast Capability Register.....	890
20-4	Multicast Control Register.....	890
20-5	Multicast Base Address Register .....	891
20-6	Position of Multicast Group Number .....	892
20-7	Multicast Address Example .....	894
20-8	Multicast Overlay BAR .....	895
20-9	Overlay Example .....	896
20-10	Device Capabilities 2 Register.....	899
20-11	TPH Example.....	901
20-12	TPH Example with System Cache.....	902
20-13	TPH Usage for TLPs to Endpoint.....	903
20-14	TPH Usage Between Endpoints.....	904
20-15	TPH Header Bits .....	905
20-16	TPH Requester Capability Structure.....	906
20-17	TPH Capability and Control Registers .....	907
20-18	TPH Capability ST Table .....	908
20-19	TPH Prefix Indication.....	909
20-20	Resizable BAR Registers .....	912
20-21	Resizable BAR Capability Register .....	912
20-22	Resizable BAR Control Register .....	913
20-23	BARs in a Type0 Configuration Header.....	914
1	LeCroy Oscilloscope with ProtoSync Software Option .....	920
2	LeCroy PCI Express Slot Interposer x16.....	922
3	LeCroy XMC, AMC, and Mini Card Interposers .....	923
4	LeCroy PCI Express Gen3 Mid-Bus Probe.....	923
5	LeCroy PCI Express Gen2 Flying Lead Probe .....	924
6	TLP Packet with ECRC Error .....	925
7	"Link Level" Groups TLP Packets with their Link Layer Response .....	925
8	"Split Level" Groups Completions with Associated Non-Posted Request.....	926
9	"Compact View" Collapses Related Packets for Easy Viewing of Link Training	927
10	LTSSM Graph Shows Link State Transitions Across the Trace .....	928
11	Flow Control Credit Tracking.....	929
12	BitTracer View of Gen2 Traffic .....	930
13	LeCroy Gen3 PETrainer Exerciser Card .....	932
14	LeCroy Gen2 Protocol Test Card (PTC) .....	933
1	MR-IOV Switch Usage .....	938
2	MR-IOV Switch Internal Architecture .....	939
3	PCIe in a Data Center for HPC Applications.....	940
4	PCIe Switch Application in an SSD Add-In Card.....	941
5	Server Motherboard Use PCIe Switches.....	941
6	Server Failover in 1 + N Failover Scheme .....	942

## **Figures**

---

1	Enumeration Using Transparent Bridges.....	947
2	Direct Address Translation .....	949
3	Look Up Table Translation Creates Multiple Windows .....	950
4	Intelligent Adapters in PCI and PCI Express Systems.....	951
5	Host Failover in PCI and PCI Express Systems.....	953
6	Dual Host in a PCI and PCI Express System .....	955
7	Dual-Star Fabric .....	957
8	Direct Address Translation .....	959
9	Lookup Table Based Translation.....	960
10	Use of Limit Register .....	961
1	Lock Sequence Begins with Memory Read Lock Request .....	967
2	Lock Completes with Memory Write Followed by Unlock Message .....	969

## **Figures**

---

---

---

# Tables

---

1	PC Architecture Book Series .....	1
1-1	Comparison of Bus Frequency, Bandwidth and Number of Slots .....	11
2-1	PCIe Aggregate Gen1, Gen2 and Gen3 Bandwidth for Various Link Widths...	43
2-2	PCI Express Request Types .....	59
2-3	PCI Express TLP Types.....	61
3-1	Enhanced Configuration Mechanism Memory-Mapped Address Range.....	98
4-1	Results of Reading the BAR after Writing All 1s To It.....	129
4-2	Results Of Reading the BAR Pair after Writing All 1s To Both .....	132
4-3	Results Of Reading the IO BAR after Writing All 1s To It.....	134
4-4	Example Prefetchable Memory Base/Limit Register Meanings.....	139
4-5	Example Non-Prefetchable Memory Base/Limit Register Meanings.....	141
4-6	Example IO Base/Limit Register Meanings .....	143
4-7	PCI Express TLP Types And Routing Methods .....	147
4-8	Posted and Non-Posted Transactions .....	150
4-9	TLP Header Format and Type Field Encodings.....	153
4-10	Message Request Header Type Field Usage.....	165
5-1	TLP Header Type Field Defines Transaction Variant .....	174
5-2	Generic Header Field Summary .....	176
5-3	TLP Header Type and Format Field Encodings.....	179
5-4	IO Request Header Fields .....	186
5-5	4DW Memory Request Header Fields .....	189
5-6	Configuration Request Header Fields .....	194
5-7	Completion Header Fields .....	197
5-8	Message Request Header Fields .....	204
5-9	INTx Interrupt Signaling Message Coding.....	207
5-10	Power Management Message Coding .....	208
5-11	Error Message Coding .....	209
5-12	Unlock Message Coding .....	209
5-13	Slot Power Limit Message Coding .....	210
5-14	Vendor-Defined Message Coding .....	211
5-15	Hot Plug Message Coding .....	212
5-16	LTR Message Coding .....	213
5-17	LTR Message Coding .....	213
6-1	Required Minimum Flow Control Advertisements .....	219
6-2	Maximum Flow Control Advertisements .....	220
6-3	Gen1 Unadjusted AckNak_LATENCY_TIMER Values (Symbol Times).....	241
6-4	Gen2 Unadjusted AckNak_LATENCY_TIMER Values (Symbol Times).....	241
6-5	Gen3 Unadjusted AckNak_LATENCY_TIMER Values (Symbol Times) .....	242
8-1	Simplified Ordering Rules Table .....	289
8-2	Transactions That Can Be Reordered Due to Relaxed Ordering .....	299
9-1	DLLP Types .....	311
9-2	Ack/Nak DLLP Fields .....	313

# Tables

---

9-3	Power Management DLLP Fields.....	314
9-4	Flow Control DLLP Fields.....	315
10-1	Ack or Nak DLLP Fields.....	329
10-2	Gen1 Unadjusted Ack Transmission Latency .....	345
10-3	Gen1 Unadjusted AckNak_LATENCY_TIMER Values (Symbol Times) .....	351
10-4	Gen2 Unadjusted AckNak_LATENCY_TIMER Values (Symbol Times) .....	352
10-5	Gen3 Unadjusted AckNak_LATENCY_TIMER Values (Symbol Times) .....	352
10-6	Gen1 Unadjusted REPLAY_TIMER Values in Symbol Times .....	353
10-7	Gen2 Unadjusted REPLAY_TIMER Values in Symbol Times .....	354
10-8	Gen3 Unadjusted REPLAY_TIMER Values .....	354
11-1	Control Character Encoding and Definition.....	386
11-2	Allowable Transmitter Signal Skew .....	391
11-3	Allowable Receiver Signal Skew .....	399
12-1	PCI Express Aggregate Bandwidth for Various Link Widths .....	408
12-2	Gen3 16-bit Skip Ordered Set Encoding.....	428
12-3	Gen3 Scrambler Seed Values.....	432
12-4	Gen3 Tap Equations for Single-LFSR Scrambler.....	433
12-5	Signal Skew Parameters.....	443
13-1	Tx Preset Encodings with Coefficients and Voltage Ratios.....	478
13-2	Tx Coefficient Table.....	480
13-3	Transmitter Specs.....	489
13-4	Parameters Specific to 8.0 GT/s.....	491
13-5	Common Receiver Characteristics .....	498
14-1	Summary of TS1 Ordered Set Contents.....	514
14-2	Summary of TS2 Ordered Set Contents.....	516
14-3	Symbol Sequence 8b/10b Compliance Pattern .....	529
14-4	Second Block of 128b/130b Compliance Pattern .....	530
14-5	Third Block of 128b/130b Compliance Pattern .....	531
14-6	Symbol Sequence of 8b/10b Modified Compliance Pattern .....	532
14-7	Sequence of Compliance Tx Settings .....	535
14-8	Tx Preset Encodings .....	579
14-9	Rx Preset Hint Encodings .....	580
14-10	Conditions for Inferring Electrical Idle.....	596
15-1	Completion Code and Description .....	663
15-2	Error Message Codes and Description .....	669
15-3	Error-Related Fields in Command Register.....	675
15-4	Error-Related Fields in Status Register.....	677
15-5	Default Classification of Errors.....	679
15-6	Errors That Can Use Header Log Registers .....	695
16-1	Major Software/Hardware Elements Involved In PC PM .....	706
16-2	System PM States as Defined by the OnNow Design Initiative .....	708
16-3	OnNow Definition of Device-Level PM States.....	709

## Tables

---

16-4	Default Device Class PM States .....	710
16-5	D0 Power Management Policies .....	714
16-6	D1 Power Management Policies .....	717
16-7	D2 Power Management Policies .....	719
16-8	D3hot Power Management Policies .....	721
16-9	D3cold Power Management Policies .....	722
16-10	Description of Function State Transitions .....	723
16-11	Function State Transition Delays .....	724
16-12	The PMC Register Bit Assignments .....	725
16-13	PM Control/Status Register (PMCSR) Bit Assignments .....	728
16-14	Data Register Interpretation.....	733
16-15	Relationship Between Device and Link Power States .....	734
16-16	Link Power State Characteristics.....	735
16-17	Electrical Idle Inference Conditions .....	741
16-18	Active State Power Management Control Field Definition.....	743
17-1	INTx Message Mapping Across Virtual PCI-to-PCI Bridges .....	809
17-2	Format and Usage of Message Control Register .....	814
17-3	Format and Usage of MSI-X Message Control Register.....	823
19-1	Introduction to Major Hot-Plug Software Elements.....	852
19-2	Major Hot-Plug Hardware Elements .....	853
19-3	Behavior and Meaning of the Slot Attention Indicator .....	860
19-4	Behavior and Meaning of the Power Indicator .....	861
19-5	Slot Capability Register Fields and Descriptions .....	866
19-6	Slot Control Register Fields and Descriptions.....	869
19-7	Slot Status Register Fields and Descriptions.....	871
19-8	The Primitives .....	875
19-9	Maximum Power Consumption for System Board Expansion Slots .....	881
20-1	PH Encoding Table.....	905
20-2	ST Table Location Encoding.....	907

## **Tables**

---

---

---

## The MindShare Technology Series

The MindShare Technology series includes the books listed in Table 1.

Table 1: PC Architecture Book Series

Category	Title	Edition	ISBN
Processor Architectures	x86 Instruction Set Architecture	1st	978-0-9770878-5-3
	The Unabridged Pentium 4	1st	0-321-24656-X
	Pentium Pro and Pentium II System Architecture	2nd	0-201-30973-4
	Pentium Processor System Architecture	2nd	0-201-40992-5
	Protected Mode Software Architecture	1st	0-201-55447-X
	80486 System Architecture	3rd	0-201-40994-1
	PowerPC 601 System Architecture	1st	0-201-40990-9
Interconnect Architectures	<i>PCI Express Technology 1.x, 2.x, 3.0</i>	1st	978-0-9770878-6-0
	Universal Serial Bus System Architecture 3.0	1st	978-0-9836465-1-8
	HyperTransport 3.1 Interconnect Technology	1st	978-0-9770878-2-2
	PCI Express System Architecture	2nd	0-321-15630-7
	Universal Serial Bus System Architecture 2.0	2nd	0-201-46137-4
	HyperTransport System Architecture	1st	0-321-16845-3
	PCI-X System Architecture	1st	0-201-72682-3
	PCI System Architecture	4th	0-201-30974-2
	Firewire System Architecture: IEEE 1394a	2nd	0-201-48535-4
	EISA System Architecture	Out-of-print	0-201-40995-X
	ISA System Architecture	3rd	0-201-40996-8

# PCI Express Technology

---

Table 1: PC Architecture Book Series (Continued)

Category	Title	Edition	ISBN
Network Architecture	InfiniBand Network Architecture	1st	0-321-11765-4
Other Architectures	PCMCIA System Architecture: 16-Bit PC Cards	2nd	0-201-40991-7
	CardBus System Architecture	1st	0-201-40997-6
	Plug and Play System Architecture	1st	0-201-41013-3
	AGP System Architecture	1st	0-201-37964-3
Storage Technologies	SAS Storage Architecture	1st	978-0-9770878-0-8
	SATA Storage Technology	1st	978-0-9770878-1-5

---

## Cautionary Note

Please keep in mind that MindShare's books often describe rapidly changing technologies, and that's true for PCI Express as well. This book is a "snapshot" of the state of the technology at the time the book was completed. We make every effort to produce the books on a timely basis, but the next revision of the spec doesn't always arrive in time to be included in a book. This PCI Express book comprehends revision 3.0 of the *PCI Express™ Base Specification* released and trademarked by the PCISIG (PCI Special Interest Group).

---

## Intended Audience

The intended audience for this book is hardware and software design, verification, and other support personnel. The tutorial approach taken may also make it useful to technical people who aren't directly involved.

---

## Prerequisite Knowledge

To get the full benefit of this material, it's recommended that the reader have a reasonable background in PC architecture, including knowledge of an I/O bus and its related protocol. Because PCI Express maintains several levels of compatibility with the original PCI design, critical background information regarding PCI has been incorporated into this book. However, the reader may well find it beneficial to read the MindShare book *PCI System Architecture*.

---

## Book Topics and Organization

Topics covered in this book and chapter flow are as follows:

**Part 1: Big Picture.** Provides an architectural perspective of the PCI Express technology by comparing and contrasting it with PCI and PCI-X buses. It also introduces features of the PCI Express architecture. An overview of configuration space concepts plus methods of packet routing are described.

**Part 2: Transaction Layer.** Includes high-level packet (TLP) format and field definitions, along with Transaction Layer functions and responsibilities such as Quality of Service, Flow Control and Transaction Ordering.

**Part 3: Data Link Layer.** Includes description of ACK/NAK error detection and correction mechanism of the Data Link Layer. DLLP format is also described.

**Part 4: Physical Layer.** Describes Lane management functions, as well as link training and initialization, reset, electrical signaling, and logical Physical Layer responsibilities associated with Gen1, Gen2 and Gen3 PCI Express.

**Part 5: Additional System Topics.** Discusses additional system topics of PCI Express, including error detection and handling, power management, interrupt handling, Hot Plug and Power Budgeting details. Additional changes made in the PCI Express 2.1 spec not described in earlier chapters are covered here.

**Part 6: Appendices.**

- 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
- Glossary

---

## Documentation Conventions

This section defines the typographical convention used throughout this book.

---

### PCI Express™

PCI Express™ is a trademark of the PCI SIG, commonly abbreviated as “PCIe”.

# **PCI Express Technology**

---

---

## **Hexadecimal Notation**

All hex numbers are followed by a lower case “h.” For example:

89F2BD02h  
0111h

---

## **Binary Notation**

All binary numbers are followed by a lower case “b.” For example:

1000 1001 1111 0010b  
01b

---

## **Decimal Notation**

Number without any suffix are decimal. When required for clarity, decimal numbers are followed by a lower case “d.” Examples:

9  
15  
512d

---

## **Bits, Bytes and Transfers Notation**

This book represents bits with a lower-case “b” and bytes with an upper-case “B.” For example:

Megabits/second = Mb/s  
Megabytes/second = MB/s  
Megatransfers/second = MT/s

---

## **Bit Fields**

Groups bits are represented with the high-order bits first followed by the low-order bits and enclosed by brackets. For example:

[7:0] = bits 0 through 7

---

---

## Active Signal States

Signals that are active low are followed by #, as in PERST# and WAKE#. Active high signals have no suffix, such as POWERGOOD.

---

## Visit Our Web Site

Our web site, [www.mindshare.com](http://www.mindshare.com), lists all of our current courses and the delivery options available for each course:

- eLearning modules
- Live web-delivered classes
- Live on-site classes.

In addition, other items are available on our site:

- Free short courses on selected topics
- Technical papers
- Errata for our books

Our books can be ordered in hard copy or eBook versions.

---

## We Want Your Feedback

MindShare values your comments and suggestions. Contact us at:

[www.mindshare.com](http://www.mindshare.com)

**Phone:** US 1-800-633-1440, International 1-575-373-0336

**General information:** [training@mindshare.com](mailto:training@mindshare.com)

**Corporate Mailing Address:**

MindShare, Inc.  
481 Highway 105  
Suite B, # 246  
Monument, CO 80132  
USA

# **PCI Express Technology**

---

---

# Part One:

# The Big Picture



---

# 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 Express Technology**

---

## **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-

# Chapter 1: Background

---

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 signalling 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

---

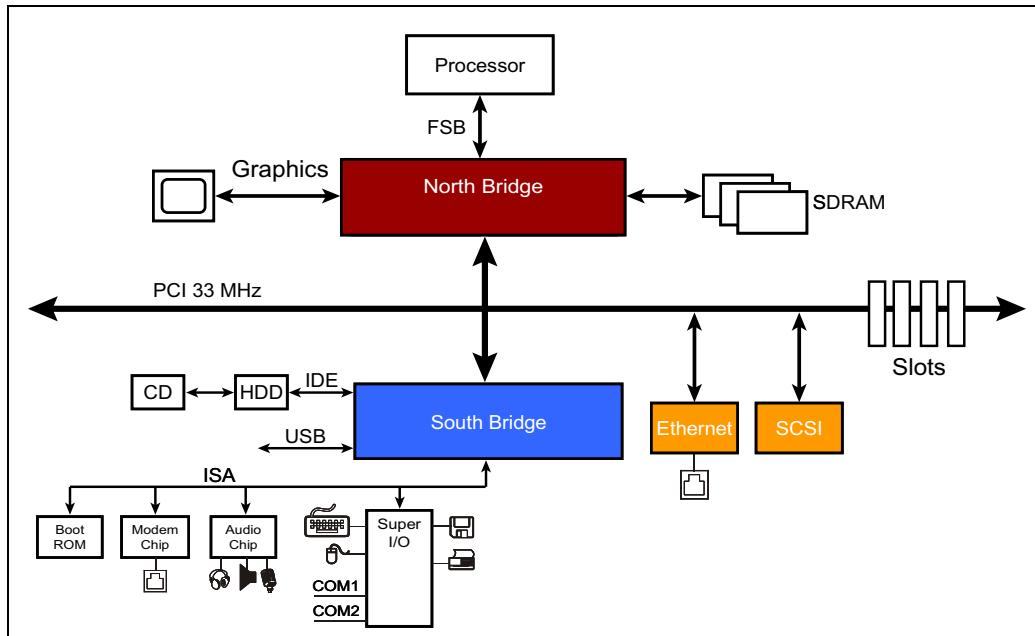
### Basics of a PCI-Based System

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.

# PCI Express Technology

---

Figure 1-1: Legacy PCI Bus-Based Platform

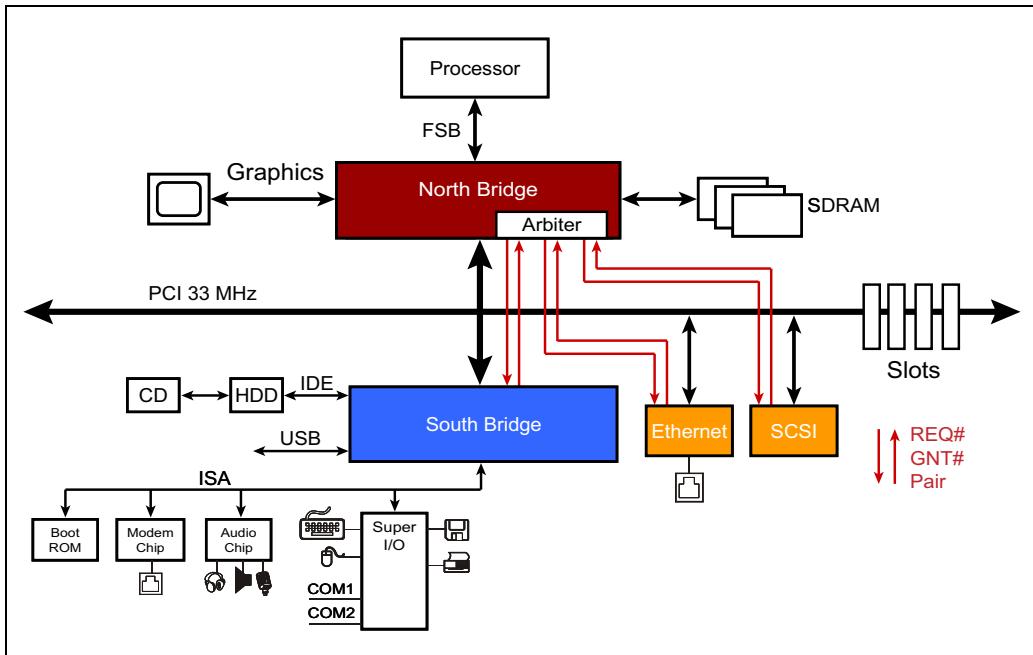


---

## 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.

Figure 1-2: PCI Bus Arbitration



## Typical PCI Bus Cycle

Figure 1-3 on page 15 represents a typical PCI bus cycle. PCI is synchronous, meaning events happen on clock edges, so the clock is shown at the top of the diagram and its rising edges are marked with dotted lines because those are the times when signals are driven out or sampled. A brief description of what happens on the bus is as follows:

1. On clock edge 1, FRAME# (used to indicate when a bus access is in progress) and IRDY# (Initiator Ready for data) are both inactive, showing that the bus is idle. At the same time, GNT# is active, meaning the bus arbiter has selected this device to be the next initiator.
2. On clock edge 2, FRAME# is asserted by the initiator, indicating that a new transaction has started. At the same time, it drives the address and command for this transaction. All of the other devices on the bus will latch this information and begin the process of decoding the address to see whether it's a match for them.

# PCI Express Technology

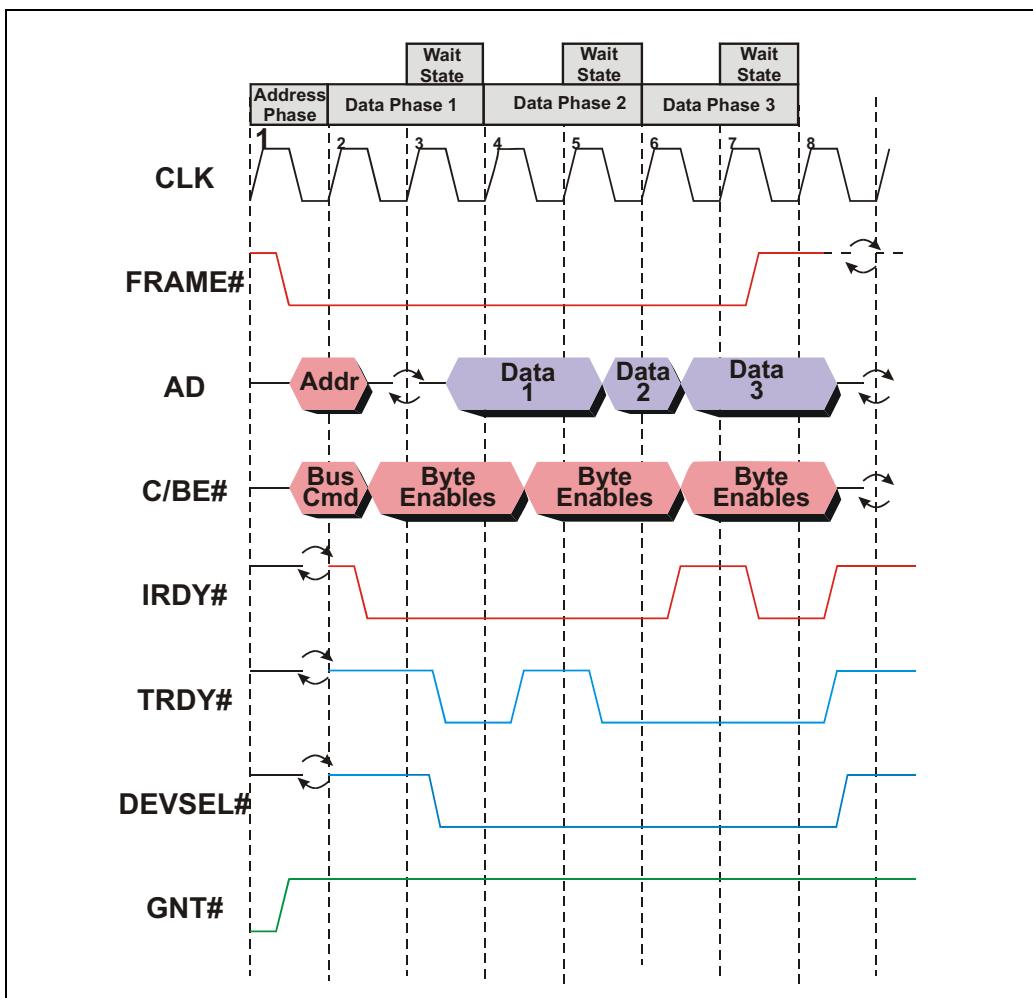
---

3. On clock edge 3, the initiator indicates its readiness for data transfer by asserting IRDY#. The round arrow symbol shown on the AD bus indicates that the tri-stated bus is undergoing a “turn-around cycle” as ownership of the signals changes (needed here because this is a read transaction; the initiator drives the address but receives data on the same pins). The target’s buffer is not turned on using the same clock edge that turns the initiator’s buffer off because we want to avoid the possibility of both buffers trying to drive a signal simultaneously, even for a brief time. That contention on the bus could damage the devices so, instead, the previous buffer is turned off one clock before the new one is turned on. Every shared signal is handled this way before changing direction.
4. On clock edge 4, a device on the bus has recognized the requested address and responded by asserting DEVSEL# (device select) to claim this transaction and participate in it. At the same time, it asserts TRDY# (target ready) to show that it is delivering the first part of the read data and drives that data onto the AD bus (this could have been delayed - the target is allowed 16 clocks from the assertion of FRAME# until TRDY#). Since both IRDY# and TRDY# are active at the same time here, data will be transferred on that clock edge, completing the first data phase. The initiator knows how many bytes will eventually be transferred, but the target does not. The command does not provide a byte count, so the target must look at the status of FRAME# whenever a data phase completes to learn when the initiator is satisfied with the amount of data transferred. If FRAME# is still asserted, this was not the last data phase and the transaction will continue with the next contiguous set of bytes, as is the case here.
5. On clock edge 5, the target is not prepared to deliver the next set of data, so it deasserts TRDY#. This is called inserting a “Wait State” and the transaction is delayed for a clock. Both initiator and target are allowed to do this, and each can delay the next data transfer by up to 8 consecutive clocks.
6. On clock edge 6, the second data item is transferred, and since FRAME# is still asserted, the target knows that the initiator still wants more data.
7. On clock edge 7, the initiator forces a Wait State. Wait States allow devices to pause a transaction to quickly fill or empty a buffer and can be helpful because they allow the transaction to resume without having to stop and restart. On the other hand, they are often very inefficient because they not only stall the current transaction, they also prevent other devices from gaining access to the bus while it’s stalled.
8. On clock edge 8, the third data set is transferred and now FRAME# has been deasserted so the target can tell that this was the last data item. Consequently, after this clock, all the control lines are turned off and the bus once again goes to the idle state.

# Chapter 1: Background

In keeping with the low cost design goal for PCI, several signals have more than one meaning on the bus to reduce the pin count. The 32 address and data signals are multiplexed and the C/BE# (Command/Byte Enable) signals share their four pins for the same reason. Although reducing the pin count is desirable, it's also the reason that PCI uses "turn-around cycles", which add more delay. It also precludes the option to pipeline transactions (sending the address for the next cycle while data for the previous one is delivered). Handshake signals like FRAME#, DEVSEL#, TRDY#, IRDY#, and STOP# control the timing of events during the transaction.

Figure 1-3: Simple PCI Bus Transfer



## Reflected-Wave Signaling

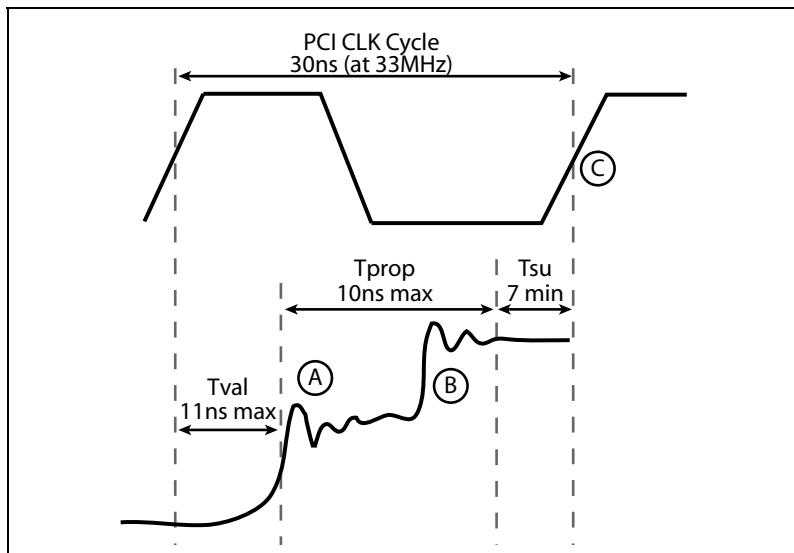
PCI architecturally supports up to 32 devices on each bus, but the practical electrical limit is considerably less, on the order of 10 to 12 electrical loads at the base frequency of 33MHz. The reason for this is that the bus uses a technique called “reflected-wave signaling” to reduce the power consumption on the bus (see Figure 1-4 on page 17). In this model, devices save cost and power by implementing weak transmit buffers that can only drive the signal to about half the voltage needed to switch the signal. The incident wave of the signal propagates down the transmission line until it reaches the end. By design, there is no termination at the end of the line so the waveform encounters an infinite impedance and reflects back. This reflection is additive in nature and increases the signal to the full voltage level as it makes its way back to the transmitter. When the signal reaches the originating buffer, the low output impedance of the driver terminates the signal and prevents further reflections. The total elapsed time from the buffer asserting a signal until the receiver detects a valid signal is thus the propagation time down the wire plus the reflection delay coming back and the setup time. All of that must be less than the clock period.

As the length of the trace and the number of electrical loads on a bus increase, the time required for the signal to make this round trip increases. A 33 MHz PCI bus can only meet the signal timing with about 10-12 electrical loads. An electrical load is one device installed on the system board, but a populated connector slot actually counts as two loads. Therefore, as indicated in Table 1-1 on page 11, a 33 MHz PCI bus can only be designed for reliable operation with a maximum of 4 or 5 add-in card connectors.

# Chapter 1: Background

---

Figure 1-4: PCI Reflected-Wave Signaling



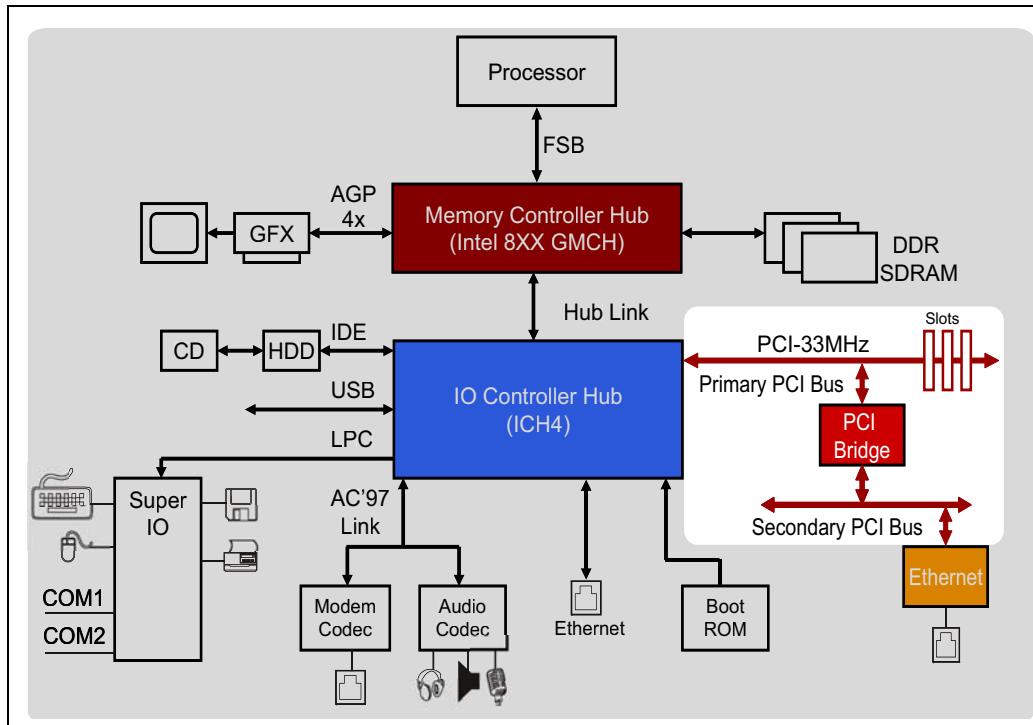
To connect more loads in a system, a PCI-to-PCI bridge is needed, as shown in Figure 1-5. By the time more modern chipsets were available, peripherals had grown so fast that their competition for access to the shared PCI bus was limiting their performance. PCI speeds didn't keep up, and it became a system bottleneck even though it was still very popular for peripherals. The solution to this problem was to move PCI out of the main path between system peripherals and memory, replacing the chipset interconnect with a proprietary solution (in this example, Intel's Hub Link interface).

A PCI Bridge is an extension to the topology. Each Bridge creates a new PCI bus that is electrically isolated from the bus above it, allowing another 10-12 loads. Some of these devices could also be bridges, allowing a large number of devices to be connected in a system. The PCI architecture allows up to 256 buses in a single system and each of those buses can have up to 32 devices.

# PCI Express Technology

---

Figure 1-5: 33 MHz PCI System, Including a PCI-to-PCI Bridge



---

## PCI Bus Architecture Perspective

### PCI Transaction Models

PCI uses three models for data transfer just as previous bus models did: Programmed I/O (PIO), Peer-to-peer, and DMA. These models are illustrated in Figure 1-6 on page 19 and described in the following sections.

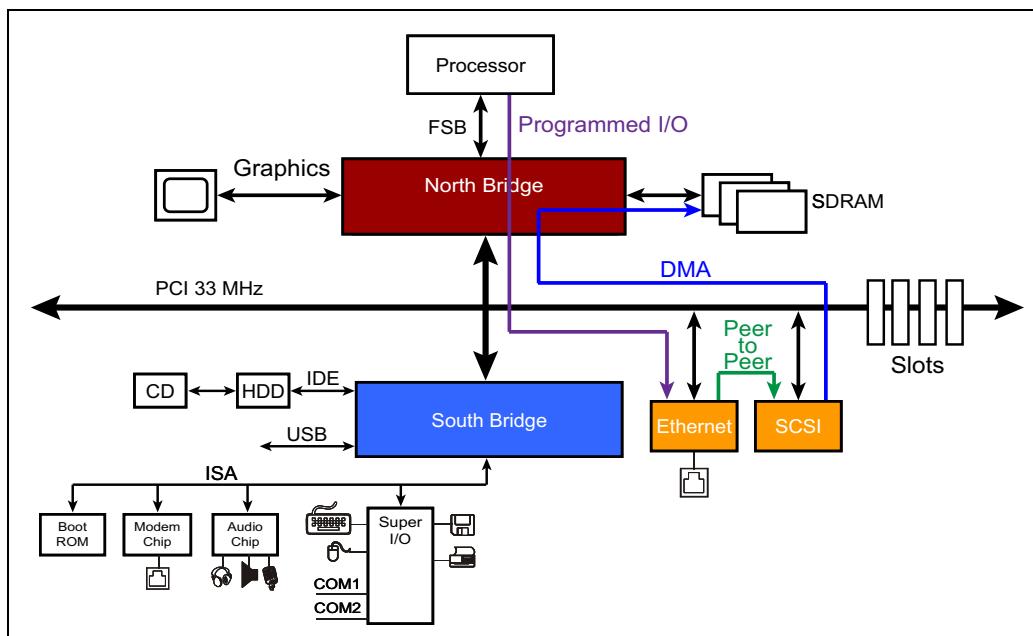
#### Programmed I/O

PIO was commonly used in the early days of the PC because designers were reluctant to add the expense or complexity to their devices of transaction management logic. The processor could do the job faster than any other device anyway so, in this model, it handles all the work. For example, if a PCI device

interrupts the CPU to indicate that it needs to put data in memory, the CPU will end up reading data from the PCI device into an internal register and then copying that register to memory. Going the other way, if data is to be moved from memory to the PCI device, software instructs the CPU to read from memory into its internal register and then write that register to the PCI device.

The process works but is inefficient for two reasons. First, there are two bus cycles generated by the CPU for every data transfer, and second, the CPU is busy with data transfer housekeeping rather than more interesting work. In the early days this was the fastest transfer method and the single-tasking processor didn't have much else to do. These types of inefficiencies are typically not acceptable in modern systems, so this method is no longer very common for data transfers, and instead the DMA method described in the next section is the preferred approach. However, programmed IO is still a necessary transaction model in order for software to interact with a device.

Figure 1-6: PCI Transaction Models



## Direct Memory Access (DMA)

A more efficient method of transferring data is called DMA (direct memory access). In this model another device, called a DMA engine, handles the details of memory transfers to a peripheral on behalf of the processor, off-loading this

# PCI Express Technology

---

tedious task. Once the CPU has programmed the starting address and byte count into it, the DMA engine handled the bus protocol and address sequencing on its own. This didn't involve any change to the PCI peripherals and allowed them to keep their low-cost designs. Later, improved integration allowed peripherals to integrate this DMA functionality locally, so they didn't need an external DMA engine. These devices were capable of handling their own bus transfers and were called Bus Master devices.

Figure 1-3 on page 15 is an example of a Bus Master transaction on PCI. The North Bridge might decode the address and recognize that it will be the target for the transaction. In the data phase of the bus cycle, data is transferred between the Bus Master and the North Bridge acting as the target. The North Bridge in turn will generate DRAM bus cycles to communicate with system memory. After the transfer is completed, the PCI peripheral might generate an interrupt to inform the system. The DMA method of data transfer is more efficient because the CPU is not involved in the data movement, and a single bus cycle may be sufficient to move a block of data.

## Peer-to-Peer

If a device is capable of acting as a Bus Master, then another interesting option presents itself. One PCI Bus Master could initiate a transfer to another PCI device, with the result that the entire transaction remains local to the PCI bus and doesn't involve any other system resources. Since this transaction takes place between devices that are considered peers in the system, it's referred to as a peer-to-peer transaction. This has some obvious efficiencies because the rest of the system remains free to do other work. Nevertheless, it's rarely used in practice because the initiator and target don't often use the same format for the data unless both are made by the same vendor. Consequently, the data usually must first be sent to memory where the CPU can reformat it before it is then transferred to the target, defeating the goal of a peer-to-peer transfer.

---

## PCI Bus Arbitration

Consider Figure 1-2 on page 13. Since PCI devices today are almost all capable of being bus-master, they are able to do both DMA and peer-to-peer transfers. In a shared bus architecture like PCI, they have to take turns on the bus, so a device that wants to initiate transactions must first request ownership of the bus from the bus arbiter. The arbiter sees all the current requests and uses an implementation-specific algorithm to decide which Bus Master gets ownership of the bus next. The PCI spec doesn't describe this algorithm, but does state that it must be "fair" and not starve any device for access.

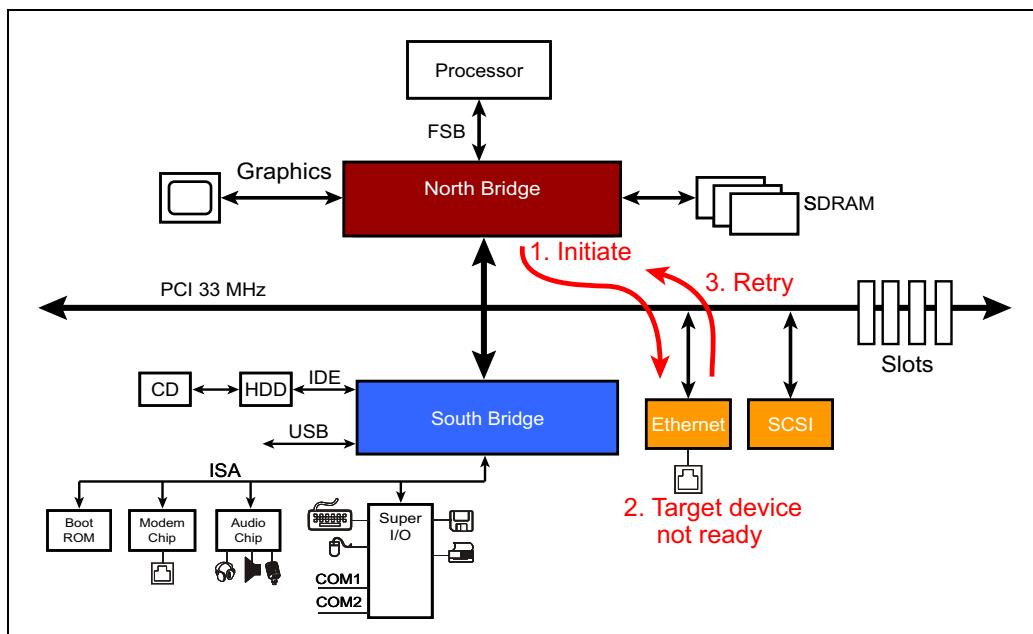
The arbiter can grant bus ownership to the next requesting device while the previous Bus Master is still executing its transfer, so that no clocks are used on the bus to sort out the next owner. As a result, the arbitration appears to happen behind the scenes and is referred to as "hidden" bus arbitration, which was a design improvement over earlier bus protocols.

## PCI Inefficiencies

### PCI Retry Protocol

When a PCI master initiates a transaction to access a target device and the target device is not ready, the target signals a transaction retry. This scenario is shown in Figure 1-7.

Figure 1-7: PCI Transaction Retry Mechanism



Consider the following example in which the North bridge initiates a memory read transaction to read data from the Ethernet device. The Ethernet target claims the bus cycle. However, the Ethernet target does not immediately have the data to return to the North bridge master. The Ethernet device has two choices by which to delay the data transfer. The first is to insert wait-states in

# PCI Express Technology

---

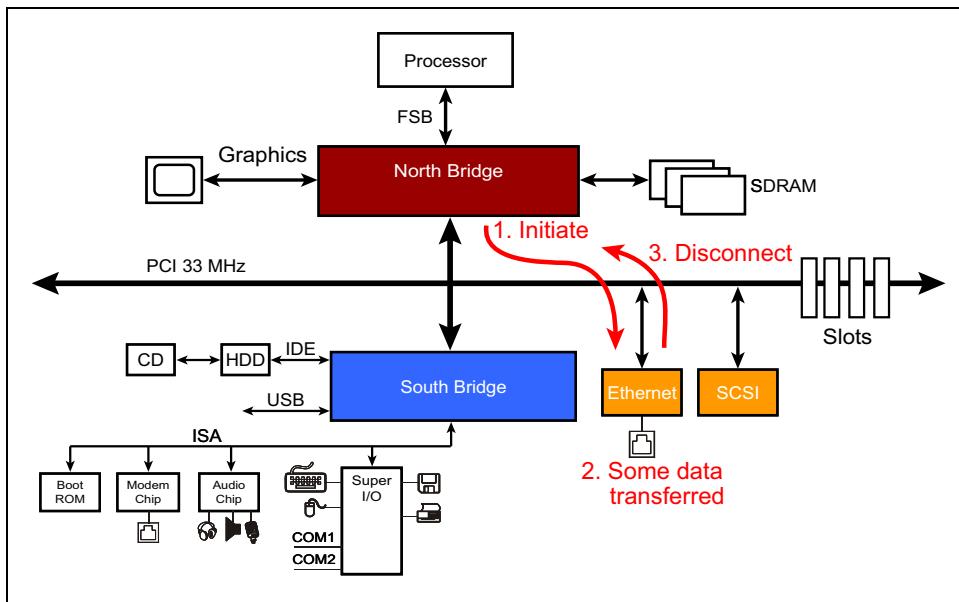
the data phase. If only a few wait-states are needed, then the data is still transferred efficiently. If however the target device requires more time (more than 16 clocks from the beginning of the transaction), then the second option the target has is to signal a retry with a signal called STOP#. A retry tells the master to end the bus cycle prematurely without transferring data. Doing so prevents the bus from being held for a long time in wait-states, which compromises the bus efficiency. The Bus Master that is retried by the target waits a minimum of 2 clocks and must once again arbitrate for use of the bus to re-initiate the identical bus cycle. During the time that the Bus Master is retried, the arbiter can grant the bus to other requesting masters so that the PCI bus is more efficiently utilized. By the time the retried master is granted the bus and it re-initiates the bus cycle, hopefully the target will claim the cycle and will be ready to transfer data. The bus cycle goes to completion with data transfer. Otherwise, if the target is still not ready, it retries the master's bus cycle again and the process is repeated until the master successfully transfers data.

## PCI Disconnect Protocol

When a PCI master initiates a transaction to access a target device and if the target device is able to transfer at least one doubleword of data but cannot complete the entire data transfer, it disconnects the transaction at the point at which it cannot continue. This scenario is illustrated in Figure 1-8 on page 23.

Consider the following example in which the North bridge initiates a burst memory read transaction to read data from the Ethernet device. The Ethernet target device claims the bus cycle and transfers some data, but then runs out of data to transfer. The Ethernet device has two choices to delay the data transfer. The first option is to insert wait-states during the current data phase while waiting for additional data to arrive. If the target needs to insert only a few wait-states, then the data is still transferred efficiently. If however the target device requires more time (the PCI specification allows maximum of 8 clocks in the data phase), then the target device must signal a disconnect. To do this the target asserts STOP# in the middle of the bus cycle to tell the master to end the bus cycle prematurely. A disconnect results in some data transferred, while a retry does not. Disconnect frees the bus from long periods of wait states. The disconnected master waits a minimum of 2 clocks before once again arbitrating for use of the bus and continuing the bus cycle at the disconnected address. During the time that the Bus Master is disconnected, the arbiter may grant the bus to other requesting masters so that the PCI bus is utilized more efficiently. By the time the disconnected master is granted the bus and continues the bus cycle, hopefully the target is ready to continue the data transfer until it is completed. Otherwise, the target once again retries or disconnects the master's bus cycle and the process is repeated until the master successfully transfers all its data.

Figure 1-8: PCI Transaction Disconnect Mechanism



---

## PCI Interrupt Handling

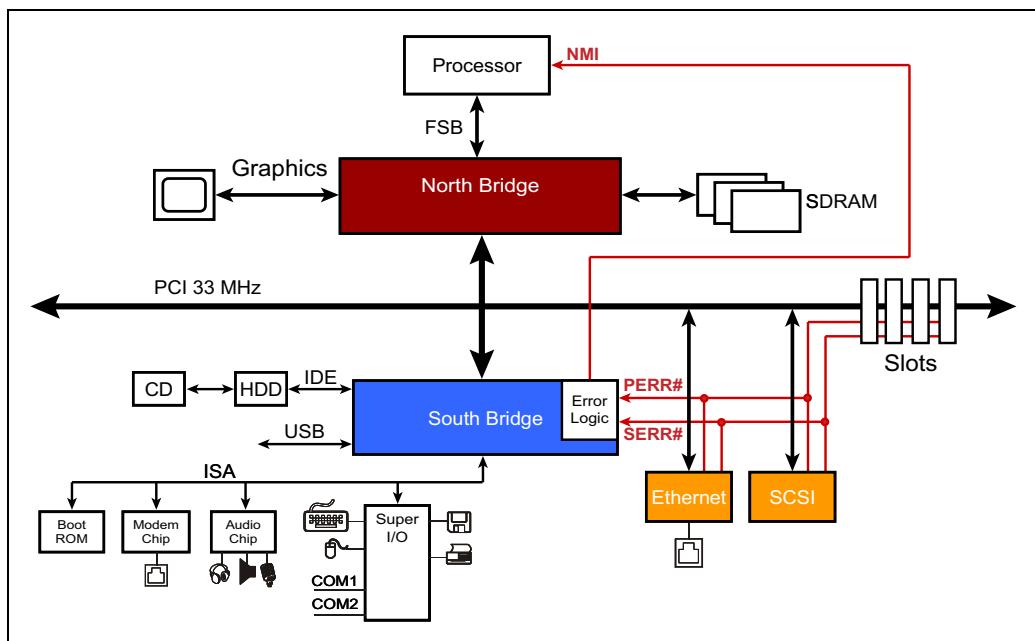
PCI devices use one of four sideband interrupt signals (INTA#, INTB#, INTC#, or INTD#) to send an interrupt request to the system. When one of the pins is asserted, the interrupt controller in a single-CPU system responded by asserting the INTR (interrupt request) pin to the CPU. Later multi-CPU designs needed to improve on the single wire input for interrupts and changed to an APIC (Advanced Programmable Interrupt Controller) model, in which the controller sends a message to the multiple CPUs instead of asserting the INTR pin to one of them. Regardless of the delivery model, an interrupted CPU must determine the source of the interrupt and then service the interrupt. The legacy model required several bus cycles for this and wasn't very efficient. The APIC model is better but also leaves room for improvement.

# PCI Express Technology

## PCI Error Handling

PCI devices can optionally detect and report address and data phase parity errors during transactions. PCI generates "even parity" across most of the signals during a transaction by using the PAR signal. This means that if the number of set bits during an address or data phase is odd, the master device will set the PAR signal to make the parity "even." The target device receives the address or data and checks for errors. Parity errors are detectable only as long as an odd number of signals are affected causing the received number of ones to be odd. If a device detects a data phase parity error, it asserts PERR# (parity error). This is potentially a recoverable error since, for cases like a memory read, just repeating the transaction may resolve the problem. PCI does not include any automatic or hardware-based recovery mechanisms, though, so any attempts to resolve the error would be handled by software.

Figure 1-9: PCI Error Handling



However, it's a different matter if a parity error is detected during the address phase. In this case the address was corrupted and the wrong target may have recognized the address. There's no way to tell what the corrupted address became or what devices on the bus did in response to it, so there's also no sim-

ple recovery. As a result, errors of this type result in the assertion of the SERR# (system error) pin, which typically results in a call to the system error handler. In older machines, this would often halt the system as a precaution, resulting in the “blue screen of death.”

In older machines, both PERR# and SERR# were connected to the error logic in the South Bridge. For reasons of simplicity and cost, this typically resulted in the assertion of an NMI signal (non-maskable interrupt signal) to the CPU, which would often simply halt the system.

---

## PCI Address Space Map

PCI architecture supports 3 address spaces as shown in Figure 1-10 on page 26: memory, I/O and configuration address space. x86 processors can access memory and IO space directly. A PCI device maps into the processors memory address space and can either support 32 or 64 bit memory addressing. In I/O address space, PCI supports 32 bit addresses but, since x86 CPUs only used 16 bits for I/O space, many platforms limit the I/O space to 64 KB (16 bits worth).

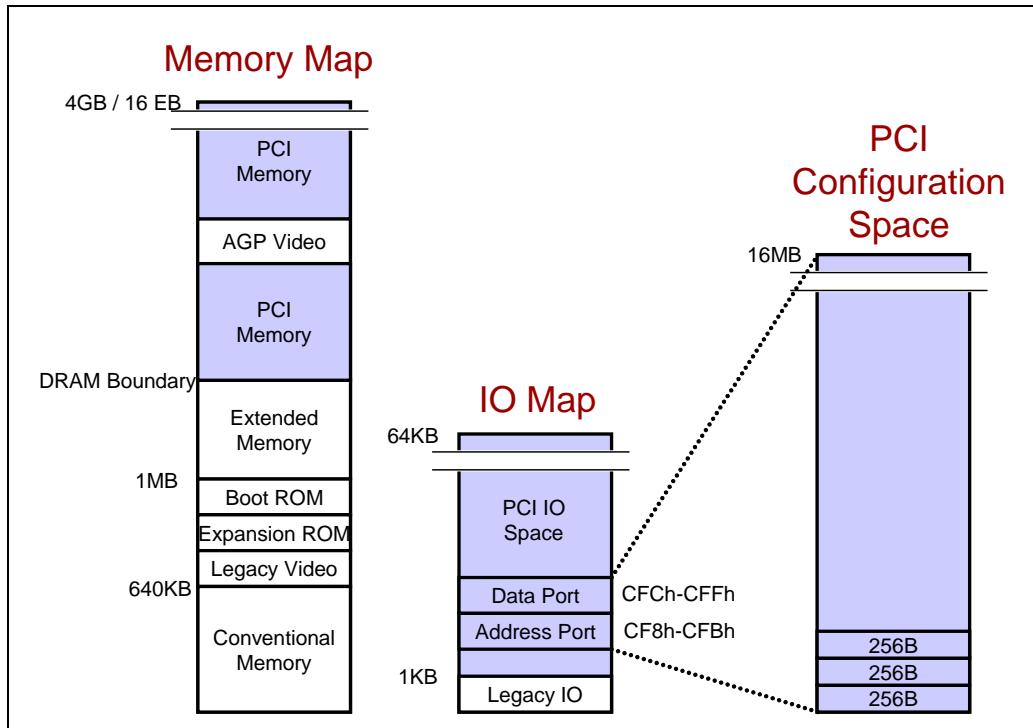
PCI also introduced a third address space called configuration space that the CPU could only indirectly access. Each function contains internal registers for configuration space that allow software visibility and control of its addresses and resources in a standardized way, providing a true “plug and play” environment in the PC. Each PCI function may have up to 256 Bytes of configuration address space. Given that PCI supports up to 8 functions/device, 32 devices/bus and up to 256 buses/system, then the total amount of configuration space associated with a system is 256 Bytes/function x 8 functions/device x 32 devices/bus x 256 buses/system = 16MB of configuration space.

Since an x86 CPU cannot access configuration space directly, it must do so indirectly by indexing through IO registers (although with PCI Express a new method to access configuration space was introduced by mapping it into the memory address space). The legacy model, shown in Figure 1-10 on page 26, uses an IO Port called Configuration Address Port located at address CF8h-CFBh and a Configuration Data Port mapped to address CFCh-CFFh. Details regarding this method and the memory mapped method of accessing configuration space are explained in the next section.

# PCI Express Technology

---

Figure 1-10: Address Space Mapping



---

## PCI Configuration Cycle Generation

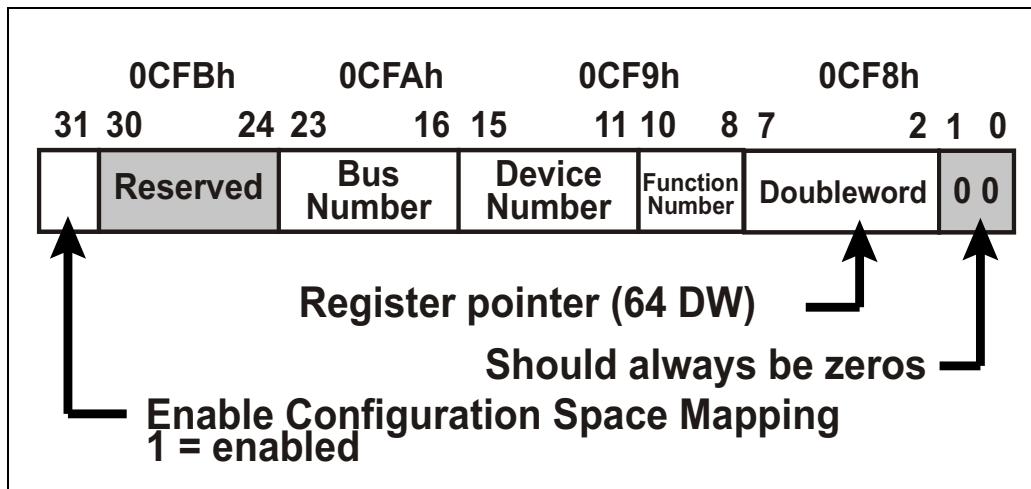
Since IO address space is limited, the legacy model was designed to be very conservative with addresses. The common way of doing that in IO space was to have one register for pointing to an internal location, and a second one for reading or writing the data. In PCI configuration that involves two steps.

Step 1: The CPU generates an IO write to the Address Port at IO address CF8h in the North Bridge to give the address of the configuration register to be accessed. This address, shown in Figure 1-11 on page 27, consists primarily of the three things that locate a PCI function within the topology: which bus we want to access out of the 256 possible, which device on that bus out of the 32 possible, and which function within that device out of the 8 possible. The only other information needed is to identify which of the 64 dwords (256 bytes) in that function's configuration space is to be accessed.

# Chapter 1: Background

Step 2: The CPU generates either an IO read or IO write to the Data Port at location CFCh in the North Bridge. Based on that, the North Bridge then generates a configuration read or configuration write transaction to the PCI bus specified in the Address Port.

Figure 1-11: Configuration Address Register



## PCI Function Configuration Register Space

Each PCI function contains up to 256 bytes of configuration space. The first 64 bytes of each function's configuration space contains a structure called the Header, while the remaining 192 Bytes support optional functionality. System configuration is first performed by Boot ROM firmware. After the OS loads, it may reconfigure the system and rearrange resource assignments, with the result that the process of system configuration may be done twice.

There are two basic classes of PCI functions as defined by their header types. A Type 1 header identifies a function that is a bridge (as shown in Figure 1-12 on page 28) and creates another bus in the topology, while a Type 0 header indicates a function that is NOT a bridge (as shown in Figure 1-13 on page 29). This header type information is contained in a field by the same name in dword 3, byte 2, and should be one of the first things software checks when discovering which functions exist in the system (a process called “enumeration”).

# PCI Express Technology

---

Figure 1-12: PCI Configuration Header Type 1 (Bridge)

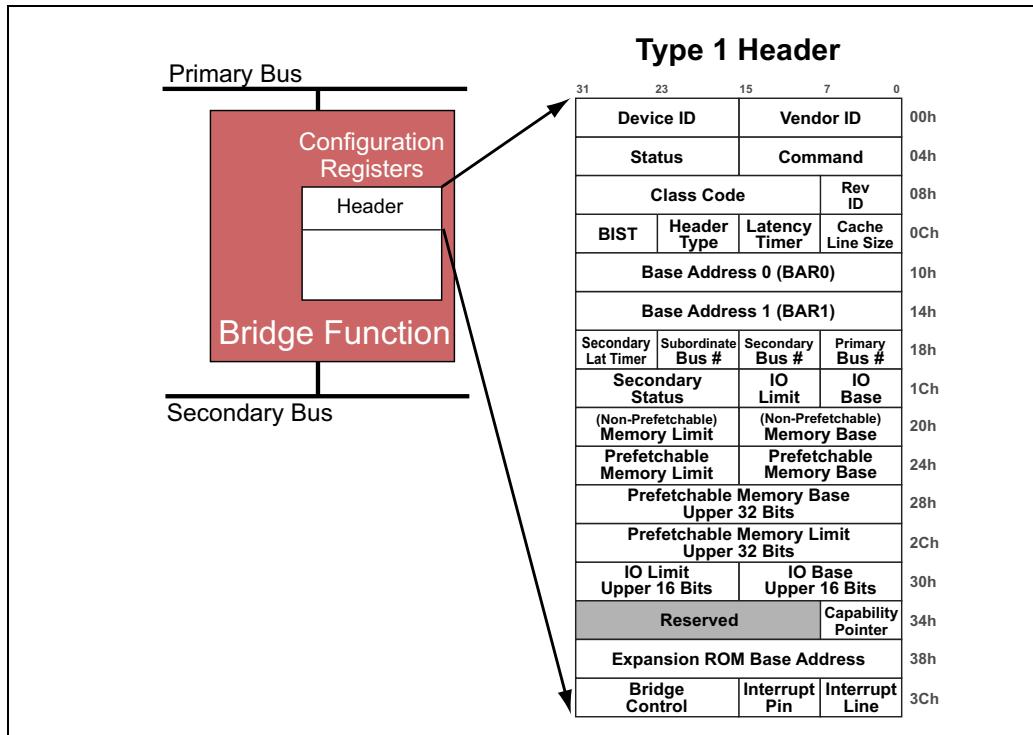
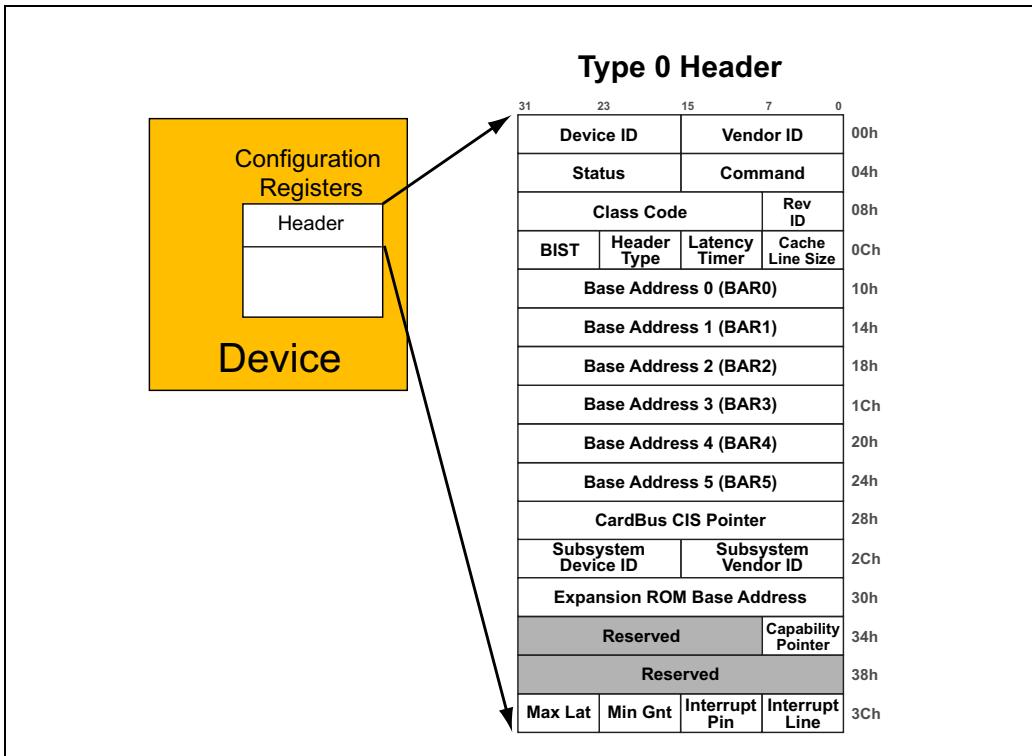


Figure 1-13: PCI Configuration Header Type 0 (not a Bridge)



Details of the configuration register space and the enumeration process are described later. For now we simply want you to become familiar with the big picture of how all the parts fit together.

---

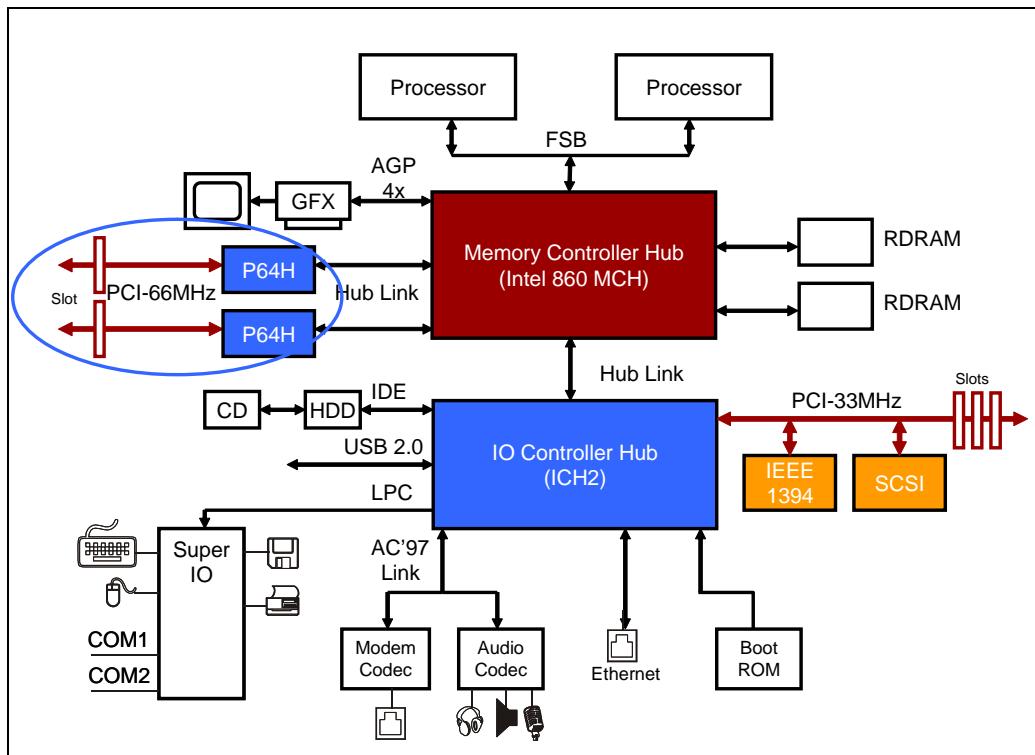
## Higher-bandwidth PCI

To support higher bandwidth, the PCI specification was updated to support both wider (64-bit) and faster (66 MHz) versions, achieving 533 MB/s. Figure 1-14 shows an example of a 66 MHz, 64-bit PCI system.

# PCI Express Technology

---

Figure 1-14: 66 MHz PCI Bus Based Platform



## Limitations of 66 MHz PCI bus

While the throughput of the bus was doubled at this speed relative to the 33 MHz bus, the diagram illustrates one of its major shortcomings: using the same reflected-wave switching model with only half the timing budget meant that the loading on the bus had to be greatly reduced. The result was that only one add-in card could be supported on each bus. Adding more device meant adding more PCI bridges and buses would increase both cost and board real estate requirements. The 64-bit PCI bus increases pin count, increasing system cost and lowered system reliability. In combination, it's easy to see why these factors limited the popularity of 64-bit or 66 MHz version of PCI bus.

## Signal Timing Problems with the Parallel PCI Bus Model beyond 66 MHz

PCI bus clock frequency cannot be increased beyond 66MHz given the realistic loads that exist on a PCI bus and signal flight times. With a 66 MHz clock, the clock period is 15 ns. Setup time allocated at the receiver is 3 ns. With the PCI “non-registered input” signal bus model, reducing signal setup time below this 3 ns value is not realistic. The rest of the 12 ns timing budget is allocated towards output delays at the transmitter and signal flight time. Clocking PCI bus any faster than 66 MHz implies reducing clock period. A transmitted signal will not be received in time enough to be sampled at the receiver.

The PCI-X bus introduced in the next section takes the approach of registering all input signals with a Flip-Flop before using them. Doing so reduced signal setup time to below 1 ns. The setup time savings of PCI setup time allows PCI-X bus to be run at higher frequencies of 100 MHz or even 133 Mhz. In the next section, we describe PCI-X bus architecture briefly.

---

## Introducing PCI-X

PCI-X is backward compatible with PCI in both hardware and software, but provides better performance and higher efficiency. It uses the same connector format, so PCI-X devices can be plugged into PCI slots and vice-versa. And it uses the same configuration model, so device drivers, operating systems, and applications that run on a PCI system also run on a PCI-X system.

To achieve higher speeds without changing the PCI signaling model, PCI-X added a few tricks to improve the bus timing. First, they implement PLL (phase-locked loop) clock generators that provide phase-shifted clocks internally. That allows the outputs to be driven a little earlier and the inputs to be sampled a little later, improving the timing on the bus. Likewise, PCI-X inputs are registered (latched) at the input pin of the target device, resulting in shorter setup times. The time gained by these means increased the time available for signal propagation on the bus and allowed higher clock frequencies.

---

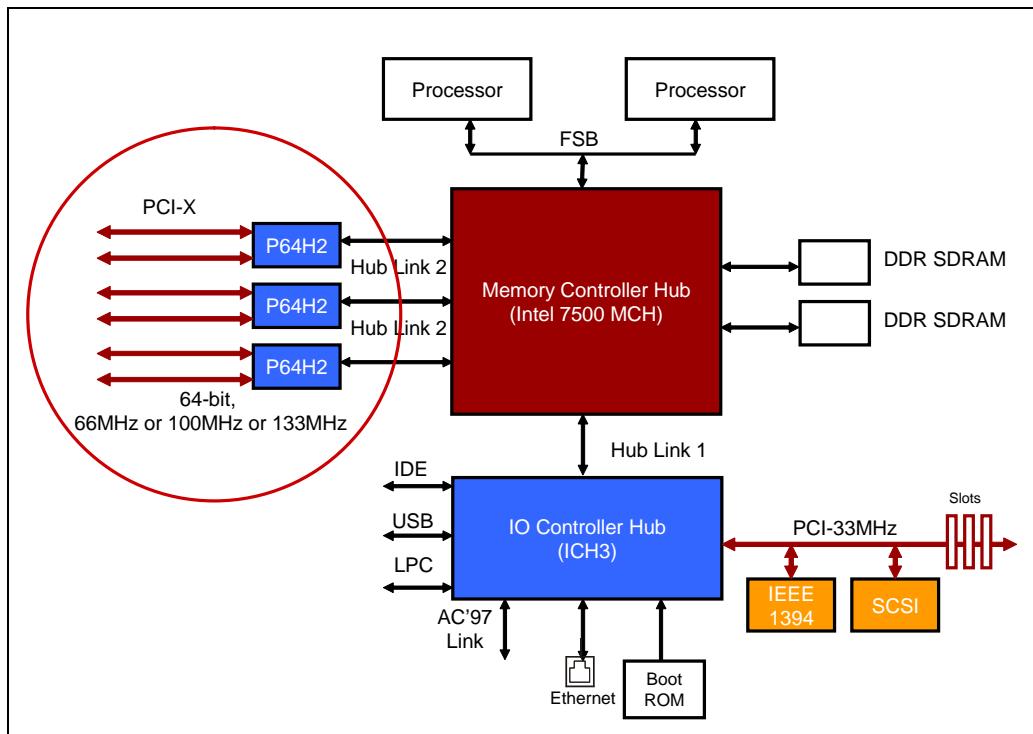
## PCI-X System Example

An example of an Intel 7500 server chipset-based system is shown in Figure 1-15 on page 32. The MCH chip includes three additional high-performance Hub Link 2.0 ports that are connected to three PCI-X Hub 2 bridges (P64H2). Each

# PCI Express Technology

bridge supports two PCI-X buses that can run at frequencies up to 133MHz. The Hub Link 2.0 can sustain the higher bandwidth requirements for PCI-X traffic. Note that we have the same loading problem that we did for 66-MHz PCI, resulting in a large number of buses needed to support more devices and a relatively expensive solution. The bandwidth is much higher now, though.

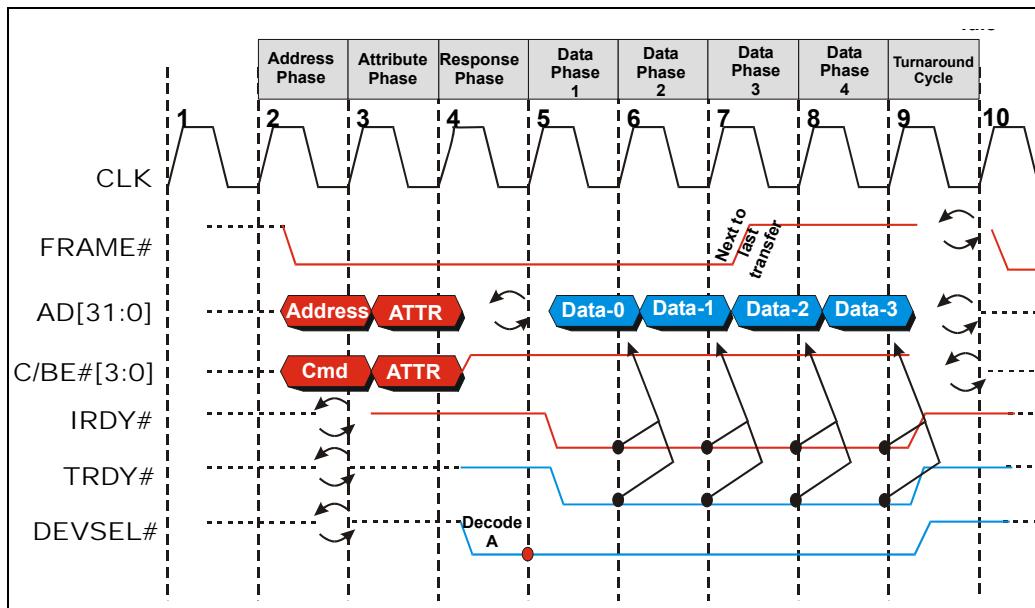
Figure 1-15: 66 MHz/133 MHz PCI-X Bus Based Platform



## PCI-X Transactions

Figure 1-16 on page 33 shows an example of a PCI-X burst memory read transaction. Note that PCI-X does not allow Wait States after the first data phase. This is possible because the transfer size is now provided to the target device in the Attribute phase of the transaction, so the target devices knows exactly what is going to be required of him. In addition, most PCI-X bus cycles are bursts and data is generally transferred in blocks of 128 Bytes. These features allow for more efficient bus utilization and device buffer management.

Figure 1-16: Example PCI-X Burst Memory Read Bus Cycle



## PCI-X Features

### Split-Transaction Model

In a conventional PCI read transaction, the Bus Master initiates a read to a target device on the bus. As described earlier, if the target is unprepared to finish the transaction it can either hold the bus with Wait States while fetching the data, or issue a Retry in the process of a Delayed Transaction.

PCI-X bus uses a Split Transaction to handle these cases, as illustrated in Figure 1-17 on page 34. To help keep track of what each device is doing, the device initiating the read is now called the Requester, and the device fulfilling the read request is called the Completer. If the completer is unable to service the request immediately, it memorizes the transaction (address, transaction type, byte count, requester ID) and signals a split response. This tells the requester to put this transaction aside in a queue, end the current bus cycle, and release the bus to the idle state. That makes the bus available for other transactions while the completer is awaiting the requested data. The requester is free to do whatever it

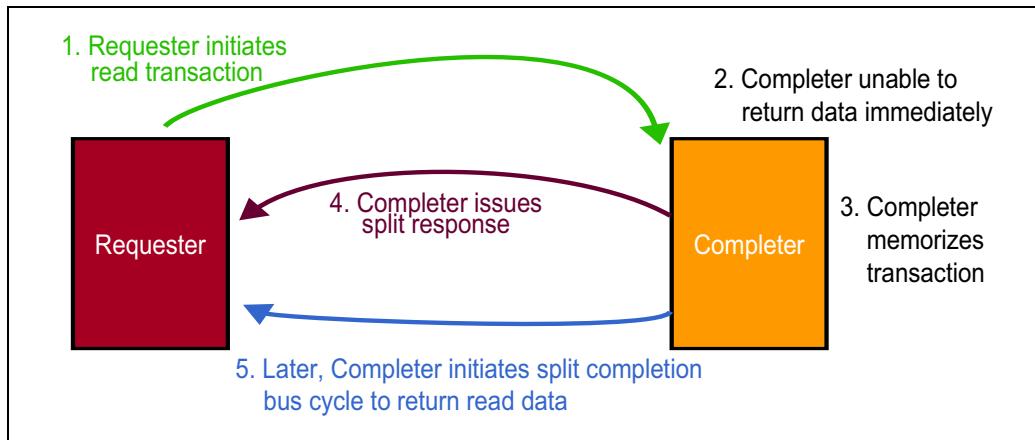
# PCI Express Technology

likes while it waits for the completer, such as initiating other requests, even to the same completer. Once the completer has gathered the requested data, it then arbitrates for ownership of the bus and initiates a split completion during which it returns the requested data. The requester claims the split completion bus cycle and accepts the data from the completer. The split completion looks very much like a write transaction to the system. This Split Transaction Model is possible because not only does the request indicate how much data they are requesting in the Attribute phase, but they also indicate who they are (their Bus:Device:Function number) which allows the completer to target the correct device with the completion.

Two bus transactions are needed to complete the entire data transfer, but between the read request and the split completion the bus is available for other work. The requester does not need to poll the device with retries to learn when the data is ready. The completer simply arbitrates for the bus and drives the requested data back when it is ready. This makes for a much more efficient transaction model in terms of bus utilization.

These protocol enhancements made to the PCI-X bus architecture described so far contribute towards an increased transfer efficiency of around 85% for PCI-X as compared to 50%-60% with the standard PCI protocol.

Figure 1-17: PCI-X Split Transaction Protocol



## Message Signaled Interrupts

PCI-X devices require MSI (Message Signaled Interrupt) capability, which was developed as a way to reduce or eliminate the need to share interrupts across multiple devices as was typically required in the legacy interrupt architecture.

To generate an interrupt request using MSI, a device initiates a memory write transaction using a pre-defined address range that is understood to be an interrupt which should be delivered to one of more CPUs, and the data is a unique interrupt vector associated with that device. The CPU, armed with the interrupt number, is able to immediately jump to the interrupt service routine for the device and avoids the overhead associated with finding which device generated the interrupt. In addition, no sideband pins are needed.

## Transaction Attributes

Finally, PCI-X also added another phase to the beginning of each transaction called the Attribute Phase (see Figure 1-16 on page 33). In this time slot the requester delivers information that can be used to help improve the efficiency of transactions on the bus, such as the byte count for this request and who the requester is (Bus:Device:Function number). In addition to those items, two new bits were added to help characterize this transaction: the "No Snoop" bit and the "Relaxed Ordering" bit.

**No Snoop (NS):** Normally, when a transaction moves data into or out of memory, the CPU's internal caches need to be checked to see if that memory location has been copied into one or more CPU caches. If so, the cache contents may need to be written back to memory or invalidated before the requested transaction is allowed to access memory. Naturally, this snoop process takes time and adds latency to a request. Sometimes the software is aware that a requested location will never be found in the CPU caches (perhaps because the location was defined by the system as uncacheable), so snooping is unnecessary and that step could be skipped. The No Snoop bit was added with precisely that case in mind.

**Relaxed Ordering (RO):** Normally, transactions are required to remain in the same order that they were issued on the bus while they go through buffers in bridges. This is referred to as the Strongly Ordered model, and PCI and PCI-X generally follow that rule with a few exceptions. That's because it helps resolve dependencies among transactions that are related to each other, such as writing and then reading the same location. However, not all transactions actually have dependencies. If they don't, then forcing them to stay in order can result in loss of performance, and that's what this bit was designed to alleviate. If the requester knows that a particular transaction is unrelated to the other transactions that have gone before, it can set this bit to tell bridges that this transaction is allowed to jump ahead in the queue to give better performance.

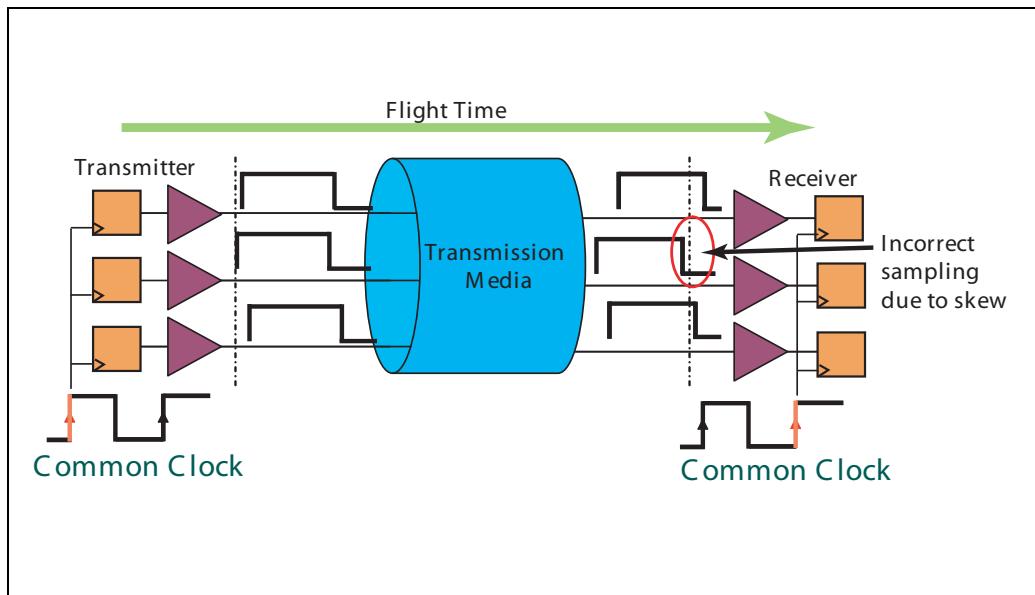
## Higher Bandwidth PCI-X

### Problems with the Common Clock Approach of PCI and PCI-X

#### 1.0 Parallel Bus Model

An issue that becomes clear when trying to migrate a bus like PCI to higher speeds is that parallel bus designs have some inherent limitations. Figure 1-18 on page 36 helps illustrate these. These designs use a common or distributed clock, in which data is driven out on one clock edge and latched in on the next clock edge so that the total timing budget is the time for one clock period. Naturally, the higher the frequency, the smaller the clock period and thus the smaller the timing budget.

Figure 1-18: Inherent Problems in a Parallel Design



The first issue to note is signal skew. When multiple data bits are sent at once, they experience slightly different delays and arrive at slightly different times at the receiver. If that difference is too large, incorrect signal sampling with clock may occur at the receiver as shown in the diagram. A second issue is clock skew between multiple devices. The arrival time of the common clock at one device is not precisely the same as the arrival time at the other which further reduces the timing budget. Finally, a third issue relates to the time it takes for the signal to

# Chapter 1: Background

---

propagate from a transmitter to a receiver, called the flight time. The clock period or timing budget must be greater than the signal flight time. To ensure this, the board design is required to implement signal traces that are short enough such that signal propagation delays are smaller than the clock period. In many board designs, this short signal traces may not be realistic enough to design for.

To further improve performance in spite of these limitations, a couple of techniques can be used. First, the existing protocol can be streamlined and made more efficient. And second, the bus model can be changed to a source synchronous clocking model where the bus signal and clock (strobe) are driven at the same time on signals that experience equal propagation delay. This is the approach taken by PCI-X 2.0 protocol.

## PCI-X 2.0 Source-Synchronous Model

PCI-X 2.0 further increased the bandwidth of PCI-X. As before, the devices and connectors remained hardware and software backward compatible with PCI devices and connectors. To achieve the higher speeds, the bus uses a source-synchronous delivery model to support either Dual Data Rate (DDR) or Quad Data Rate (QDR).

The term “source synchronous” means that the device transmitting the data also provides another signal that travels the same basic path as the data. As illustrated in Figure 1-19 on page 38, that signal in PCI-X 2.0 is called a “strobe” and is used by the receiver for latching the incoming data bits. The transmitter assigns the timing relationship between the data and strobe and as long as their paths are similar in length and other characteristics that can affect transmission latency, that relationship will be about the same when they arrive at the receiver and the receiver can simply use the Strobe as the signal to latch the data in with. This allows higher speeds because clock skew with respect to the common clock is removed as a separate budget item and because the issue of flight time goes away. It no longer matters how long it takes for the data to travel from point A to point B because the strobe that latches it in takes about the same time and so their relationship will be unaffected.

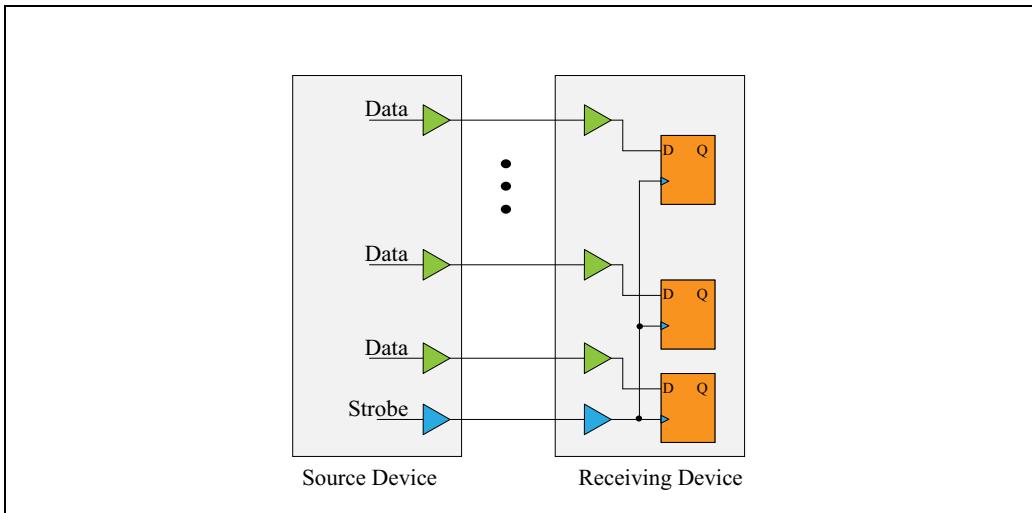
It’s important to note again that the very high-speed signal timing eliminates the possibility of using a shared-bus model and forces a point-to-point design instead. As a result, increasing the number of devices means more bridges will be needed to create more buses. A device could be designed to support this with three interfaces and an internal bridge structure to allow them all to communicate with each other. Such a device would have a very high pin count, though, and a higher cost, relegating PCI-X 2.0 to the very high-end market.

# PCI Express Technology

---

Since it was recognized that this would be an expensive solution that would appeal more to high-end designers, PCI-X 2.0 also supports ECC generation and checking. ECC is much more robust and sophisticated than parity detection, allowing automatic correction of single-bit errors on the fly, and robust detection of multi-bit errors. This improved error handling adds cost, but high-end platforms need the improved reliability it provides, hence a logical choice.

*Figure 1-19: Source-Synchronous Clocking Model*



Despite the improvements in bandwidth, efficiency and reliability that came with PCI-X (2.0), the parallel bus model was approaching its end of life and a new model was needed to address the relentless demand for higher bandwidth and lower cost. The new model chosen was a serial interface which is a drastically different bus from a physical perspective, but was still made to be software backwards compatible. We know this new model as PCI Express.

---

# 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**

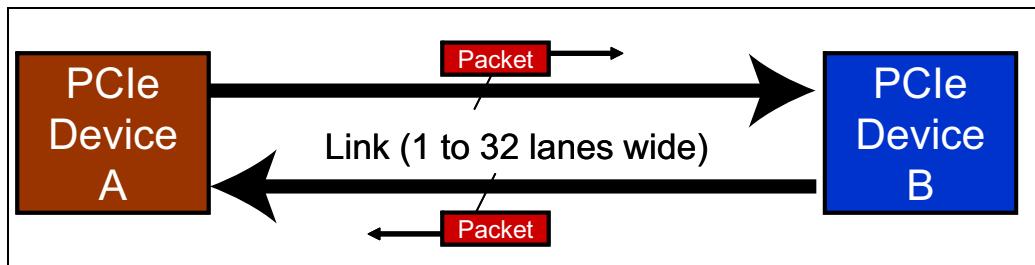
PCI Express represents a major shift from the parallel bus model of its predecessors. As a serial bus, it has more in common with earlier serial designs like InfiniBand or Fibre Channel, but it remains fully backward compatible with PCI in software.

# PCI Express Technology

---

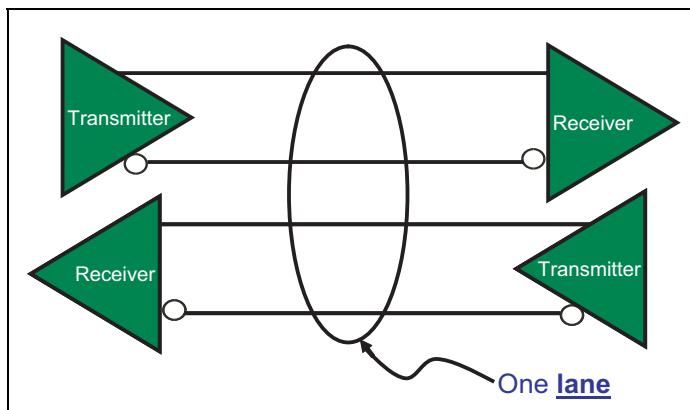
As is true of many high-speed serial transports, PCIe uses a bidirectional connection and is capable of sending and receiving information at the same time. The model used is referred to as a dual-simplex connection because each interface has a simplex transmit path and a simplex receive path, as shown 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.

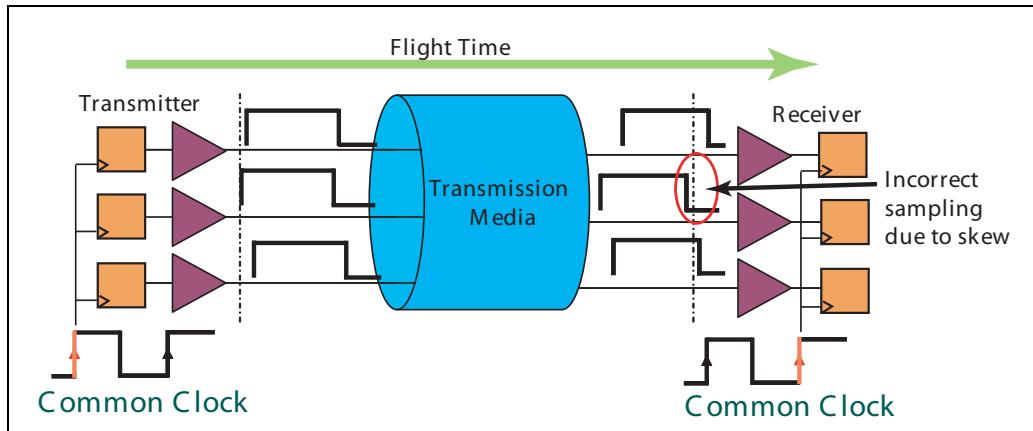
**Overcoming Problems.** By way of review, there are a handful of problems that limit the performance of a parallel bus and three are illustrated in Figure 2-3 on page 42. To get started, recall that parallel buses use a common clock; outputs are clocked out on one clock edge and clocked into the receiver 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

# PCI Express Technology

---

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

## Chapter 2: PCIe Architecture Overview

---

know that sending one byte of data requires transmitting 10 bits. The first generation (Gen1 or PCIe spec version 1.x) bit rate is 2.5 GT/s and dividing that by 10 means that one lane will be able to send 0.25 GB/s. Since the Link permits sending and receiving at the same time, the aggregate bandwidth can be twice that amount, or 0.5 GB/s per Lane. Doubling the frequency for the second generation (Gen2 or PCIe 2.x) doubled the bandwidth. The third generation (Gen3 or PCIe 3.0) doubles the bandwidth yet again, but this time the spec writers chose not to double the frequency. Instead, for reasons we'll discuss later, they chose to increase the frequency only to 8 GT/s and remove the 8b/10b encoding in favor of another encoding mechanism called **128b/130b** encoding (for more on this, see the chapter "Physical Layer - Logical (Gen3)" on page 407). Table 2-1 summarizes the bandwidth available for all the current possible combinations and shows the peak throughput the Link could deliver in that configuration.

Table 2-1: PCIe Aggregate Gen1, Gen2 and Gen3 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

### PCIe Bandwidth Calculation

To calculate the bandwidth numbers included in the table above, see the calculations outlined below.

- Gen1 PCIe Bandwidth =  $(2.5 \text{ Gb/s} \times 2 \text{ directions}) / 10 \text{ bits per symbol} = 0.5 \text{ GB/s}$ .
- Gen2 PCIe Bandwidth =  $(5.0 \text{ Gb/s} \times 2 \text{ directions}) / 10 \text{ bits per symbol} = 1.0 \text{ GB/s}$ .

Note that in the above calculations, we divide by 10 bits per symbol not 8 bits per byte, because both Gen1 and Gen2 protocols require packet bytes to be encoded using 8b/10b encoding schemes before packet transmission.

# PCI Express Technology

---

- Gen3 PCIe Bandwidth =  $(8.0 \text{ Gb/s} \times 2 \text{ directions}) / 8 \text{ bits per byte} = 2.0 \text{ GB/s}$ .

Note that at Gen3 speed, we divide by 8 bits per byte not by 10 bits per symbol because at Gen3 speed, packets are NOT 8b/10b encoded, rather they are 128b/130b encoded. There is an addition 2 bit overhead every 128 bits, but it is not large enough to account for in the calculation.

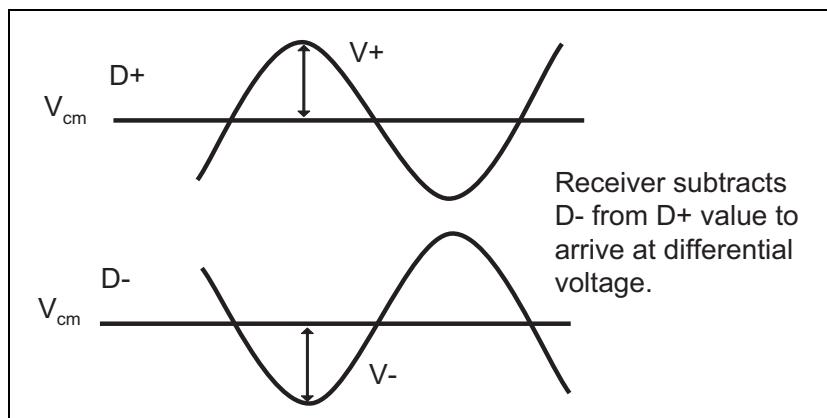
These 3 calculated bandwidth numbers are multiplied by Link width to result in total Link bandwidth on multi-Lane Links.

## Differential Signals

Each Lane uses differential signaling, sending both a positive and negative version ( $D+$  and  $D-$ ) of the same signal as shown in Figure 2-4 on page 44. This doubles the pin count, of course, but that's offset by two clear advantages over single-ended signaling that are important for high speed signals: improved noise immunity and reduced signal voltage.

The differential receiver gets both signals and subtracts the negative voltage from the positive one to find the difference between them and determine the value of the bit. Noise immunity is built in to the differential design because the paired signals are on adjacent pins of each device and their traces must also be routed very near each other to maintain the proper transmission line impedance. Consequently, anything that affects one signal will also affect the other by about the same amount and in the same direction. The receiver is looking at the difference between them and the noise doesn't really change that difference, so the result is that most noise affecting the signals doesn't affect the receiver's ability to accurately distinguish the bits.

Figure 2-4: Differential Signaling



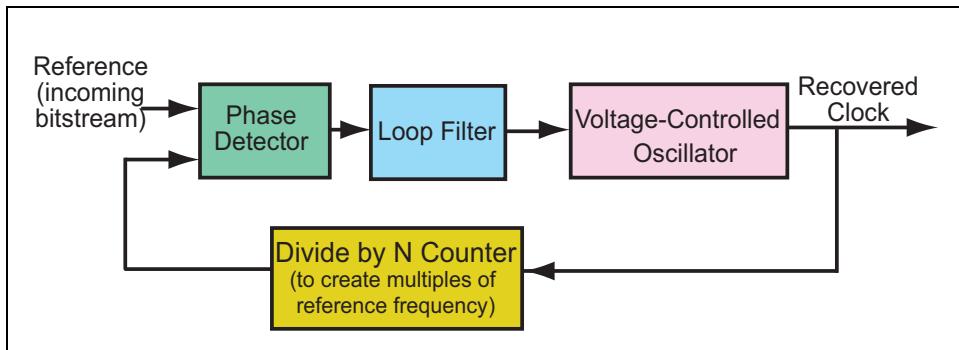
# Chapter 2: PCIe Architecture Overview

## No Common Clock

As mentioned earlier, a common clock is not required for a PCIe Link because it uses a source-synchronous model, meaning the transmitter supplies the clock to the receiver to use in latching the incoming data. A PCIe Link does not include a forwarded clock. Instead, the transmitter embeds the clock into the data stream using 8b/10b encoding. The receiver then recovers the clock from the data stream and uses it to latch the incoming data. As mysterious as this might sound, the process by which this is done is actually fairly straightforward. In the receiver, a PLL circuit (Phase-Locked Loop, see Figure 2-5 on page 45) takes the incoming bit stream as a reference clock and compares its timing, or phase, to that of an output clock that it has created with a specified frequency. Based on the result of that comparison, the output clock's frequency is increased or decreased until a match is obtained. At that point the PLL is said to be locked, and the output (recovered) clock frequency precisely matches the clock that was used to transmit the data. The PLL continually adjusts the recovered clock, so changes in temperature or voltage that affect the transmitter clock frequency will always be quickly compensated.

One thing to note regarding clock recovery is that the PLL does need transitions on the input in order to make its phase comparison. If a long time goes by without any transitions in the data, the PLL could begin to drift away from the correct frequency. To prevent that problem, one of the design goals of 8b/10b encoding is ensure no more than 5 consecutive ones or zeroes in a bit-stream (to learn more on this, refer to “8b/10b Encoding” on page 380).

Figure 2-5: Simple PLL Block Diagram



# PCI Express Technology

---

Once the clock has been recovered it's used to latch the bits of the incoming data stream into the deserializer. Sometimes students wonder whether this recovered clock can be used to clock all the logic in the receiver, but it turns out that the answer is no. One reason is that a receiver can't count on this reference always being present, because low power states on the Link involve stopping data transmission. Consequently, the receiver must also have its own internal clock that can be locally generated.

## Packet-based Protocol

Moving from a parallel to a serial transport greatly reduces the pins needed to carry data. PCIe, like most other serial-based protocols, also reduces pin count by eliminating most side-band control signals typically found in parallel buses. However, if there are no control signals indicating the type of information being received, how can the receiver interpret the incoming bits? All transactions in PCIe are sent in defined structures called packets. The receiver finds the packet boundaries and, knowing the pattern to expect, decodes the packet structure to determine what it should do.

The details of the packet-based protocol are covered in the chapter called "TLP Elements" on page 169, but an overview of the various packet types and their uses can be found in this chapter; see "Data Link Layer" on page 72.

---

## Links and Lanes

As mentioned earlier, a physical connection between two PCIe devices is called a Link and is made up of one or more Lanes. Each Lane consists of a differential send and receive signal pair, as shown in Figure 2-2 on page 40. One lane is sufficient for all communications between devices and no other signals are required.

## Scalable Performance

However, using more Lanes will increase the performance of a Link, which depends on its speed and Link width. For example, using multiple Lanes increases the number of bits that can be sent with each clock and thus improves the bandwidth. As noted earlier in Table 2-1 on page 43, the number of Lanes supported by the spec includes powers of 2 up to 32 Lanes. A x12 Link is also supported, which may have been intended to support the x12 Link width used by InfiniBand, an earlier serial design. Allowing a variety of Link widths permits a platform designer to make the appropriate trade-off between cost and performance, easily scaling up or down based on the number of Lanes.

# Chapter 2: PCIe Architecture Overview

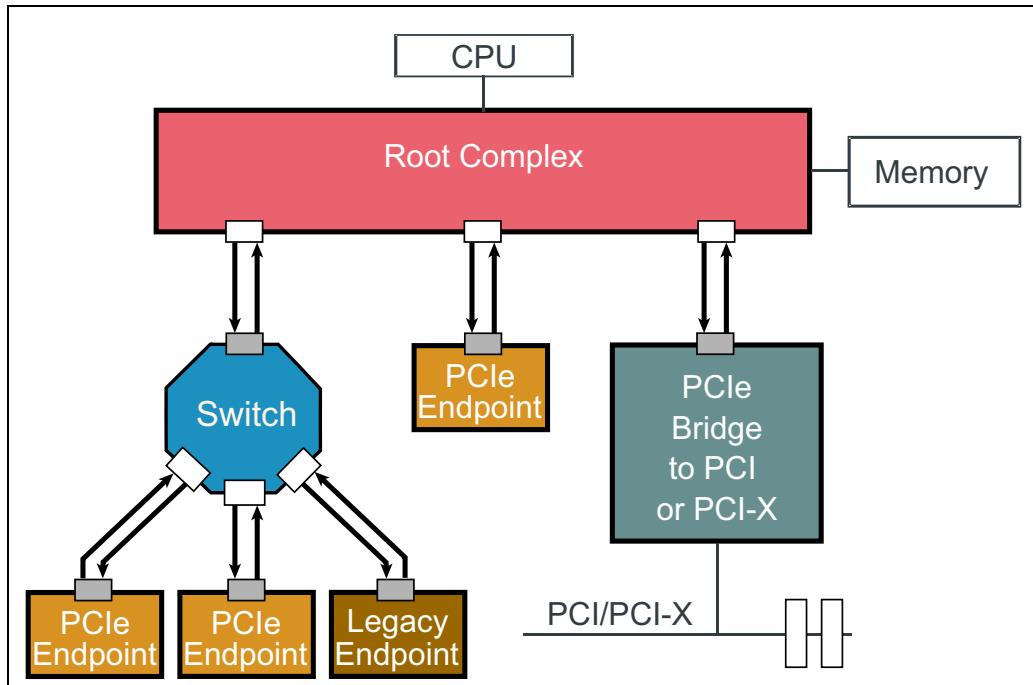
## Flexible Topology Options

A Link must be a point-to-point connection, rather than a shared bus like PCI, because of the very high speeds it uses. Since a Link can therefore only connect two interfaces, a means for fanning out the connections is needed for building a non-trivial system. This is accomplished in PCIe with the use of Switches and Bridges, which allow flexibility in constructing the system topology - the set of connections between the elements in the system. Definitions of the elements in a system and some topology examples are given in the following section.

## Some Definitions

A simple PCIe topology example is shown in Figure 2-6 on page 47, and will help illustrate some definitions at this point.

Figure 2-6: Example PCIe Topology



# PCI Express Technology

---

## Topology Characteristics

At the top of the diagram is a CPU. The point to make here is that the CPU is considered the top of the PCIe hierarchy. Just like PCI, only simple tree structures are permitted for PCIe, meaning no loops or other complex topologies are allowed. That's done to maintain backward compatibility with PCI software, which used a simple configuration scheme to track the topology and did not support complex environments.

To maintain that compatibility, software must be able to generate configuration cycles in the same way as before and the bus topology must appear the same as it did before. Consequently, all the configurations registers software expects to find are still there and behave in the same way they always have. We'll come back to this discussion a little later, after we've had a chance to define some more terms.

## Root Complex

The interface between the CPU and the PCIe buses may contain several components (processor interface, DRAM interface, etc.) and possibly even several chips. Collectively, this group is referred to as the Root Complex (RC or Root). The RC resides at the “root” of the PCI inverted tree topology and acts on behalf of the CPU to communicate with the rest of the system. The spec does not carefully define it, though, giving instead a list of required and optional functionality. In broad terms, the Root Complex can be understood as the interface between the system CPU and the PCIe topology, with PCIe Ports labeled as “Root Ports” in configuration space.

## Switches and Bridges

Switches provide a fanout or aggregation capability and allow more devices to be attached to a single PCIe Port. They act as packet routers and recognize which path a given packet will need to take based on its address or other routing information.

Bridges provide an interface to other buses, such as PCI or PCI-X, or even another PCIe bus. The bridge shown in the “Example PCIe Topology” on page 47 is sometimes called a “forward bridge” and allows an older PCI or PCI-X card to be plugged into a new system. The opposite type or “reverse bridge” allows a new PCIe card to be plugged into an old PCI system.

# Chapter 2: PCIe Architecture Overview

---

## Native PCIe Endpoints and Legacy PCIe Endpoints

Endpoints are devices in a PCIe topology that are not Switches or bridges and act as initiators and Completers of transactions on the bus. They reside at the bottom of the branches of the tree topology and only implement a single Upstream Port (facing toward the Root). By comparison, a Switch may have several Downstream Ports but can only have one Upstream Port. Devices that were designed for the operation of an older bus like PCI-X but now have a PCIe interface designate themselves as “Legacy PCIe Endpoints” in a configuration register and this topology includes one. They make use of things that are prohibited in newer PCIe designs, such as IO space and support for IO transactions or Locked requests. In contrast, “Native PCIe Endpoints” would be PCIe devices designed from scratch as opposed to adding a PCIe interface to old PCI device designs. Native PCIe Endpoints device are memory mapped devices (MMIO devices).

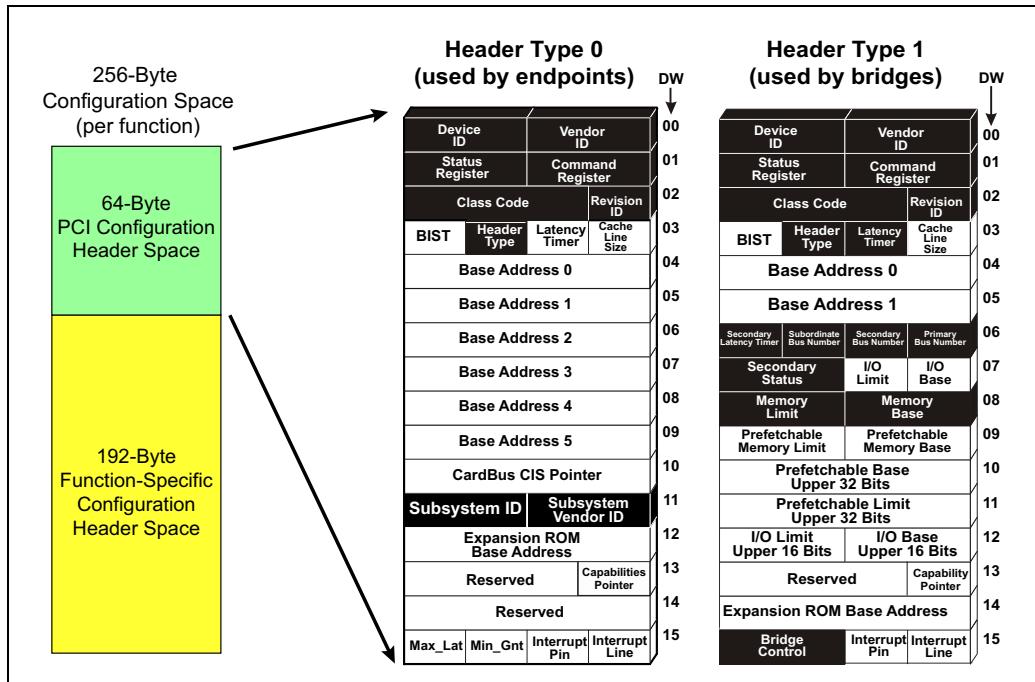
## Software Compatibility Characteristics

One way compatibility with older software is maintained is that the configuration headers for Endpoints and bridges, shown in Figure 2-7 on page 50, are unchanged from PCI. One difference now is that bridges are often aggregated into Switches and Roots, but legacy software is unaware of that distinction and will still simply see them as bridges. At this point we just want to get familiar with the concepts, so we won’t get into the details of the registers here. An introduction to the rather large topic of configuration can be found in “Configuration Overview” on page 85.

# PCI Express Technology

---

Figure 2-7: Configuration Headers

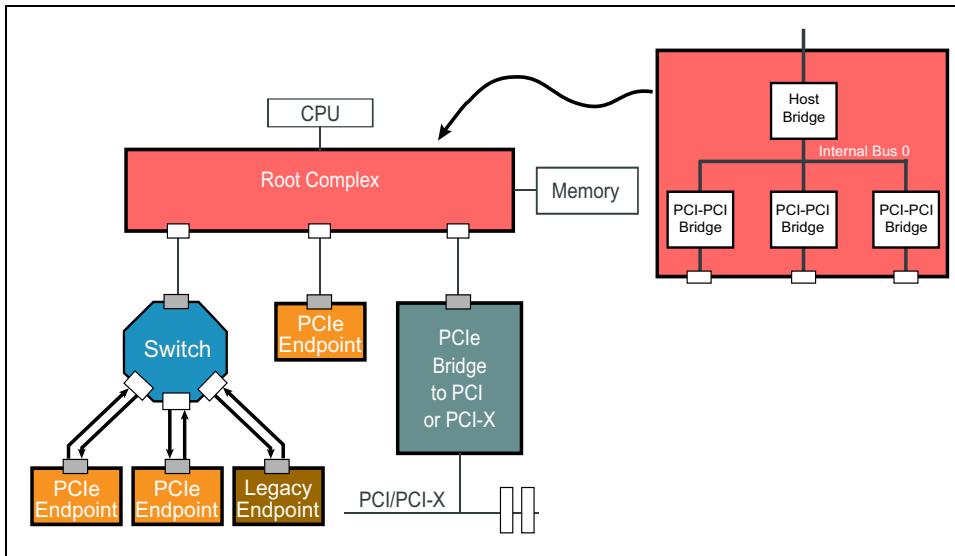


To illustrate the way the system appears to software, consider the example topology shown in Figure 2-8 on page 51. As before, the Root resides at the top of the hierarchy. The Root can be quite complex internally, but it will usually implement an internal bus structure and several bridges to fan out the topology to several ports. That internal bus will appear to configuration software as PCI bus number zero and the PCIe Ports will appear as PCI-to-PCI bridges. This internal structure is not likely to be an actual PCI bus, but it will appear that way to software for this purpose. Since this bus is internal to the Root, its actual logical design doesn't have to conform to any standard and can be vendor specific.

## Chapter 2: PCIe Architecture Overview

---

Figure 2-8: Topology Example

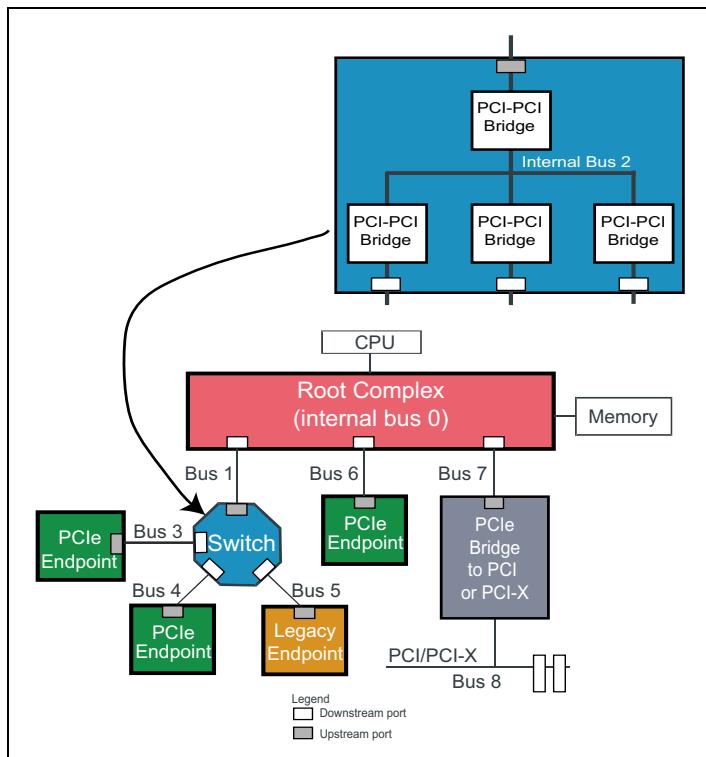


In a similar way, the internal organization of a Switch, shown in Figure 2-9 on page 52, will appear to software as simply a collection of bridges sharing a common bus. A major advantage of this approach is that it allows transaction routing to take place in the same way it did for PCI. Enumeration, the process by which configuration software discovers the system topology and assigns bus numbers and system resources, works the same way, too. We'll see some examples of how enumeration works later, but once it's been completed the bus numbers in the system will have all been assigned in a manner like that shown in Figure 2-9 on page 52.

# PCI Express Technology

---

Figure 2-9: Example Results of System Enumeration



## System Examples

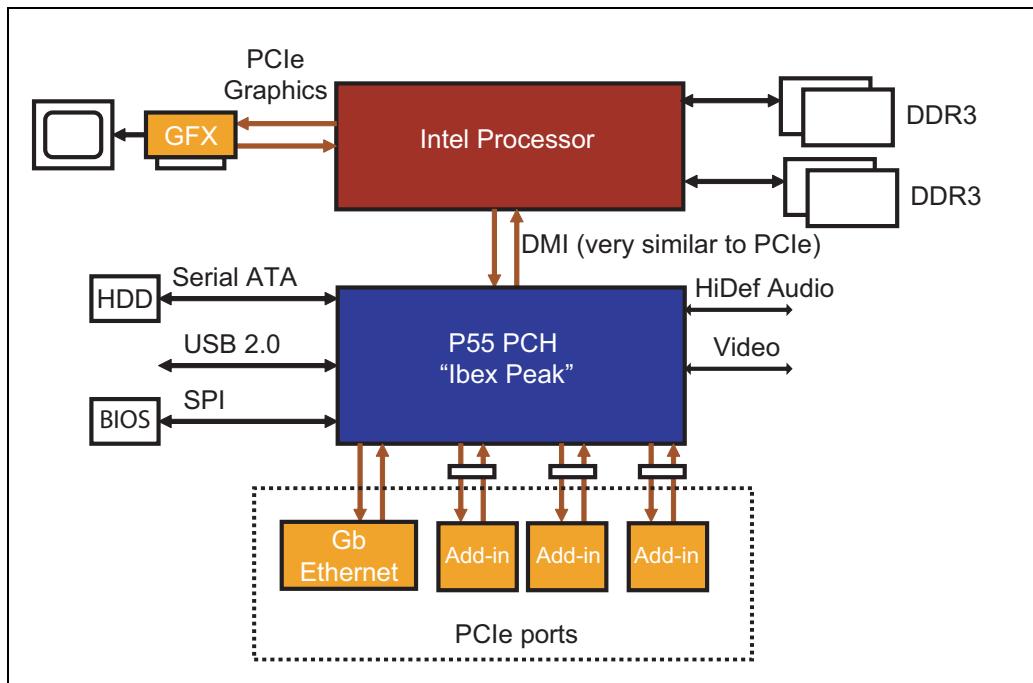
Figure 2-10 on page 53 illustrates an example of a PCIe-based system designed for a low-cost application like a consumer desktop machine. A few PCIe Ports are implemented, along with a few add-in cards slots, but the basic architecture doesn't differ much from the old-style PCI system.

By contrast, the high-end server system shown in Figure 2-11 on page 54 shows other networking interfaces built into the system. In the early days of PCIe some thought was given to making it capable of operating as a network that could replace those older models. After all, if PCIe is basically a simplified version of other networking protocols, couldn't it fill all the needs? For a variety of reasons, this concept never really achieved much momentum and PCIe-based systems still generally connect to external networks using other transports.

## Chapter 2: PCIe Architecture Overview

This also gives us an opportunity to revisit the question of what constitutes the Root Complex. In this example, the block labeled as “Intel Processor” contains a number of components, as is true of most modern CPU architectures. This one includes a x16 PCIe Port for access to graphics, and 2 DRAM channels, which means the memory controller and some routing logic has been integrated into the CPU package. Collectively, these resources are often called the “Uncore” logic to distinguish them from the several CPU cores and their associated logic in the package. Since we previously described the Root as being the interface between the CPU and the PCIe topology, that means that part of the Root must be inside the CPU package. As shown by the dashed line in Figure 2-11 on page 54, the Root here consists of part of several packages. This will likely be the case for many future system designs.

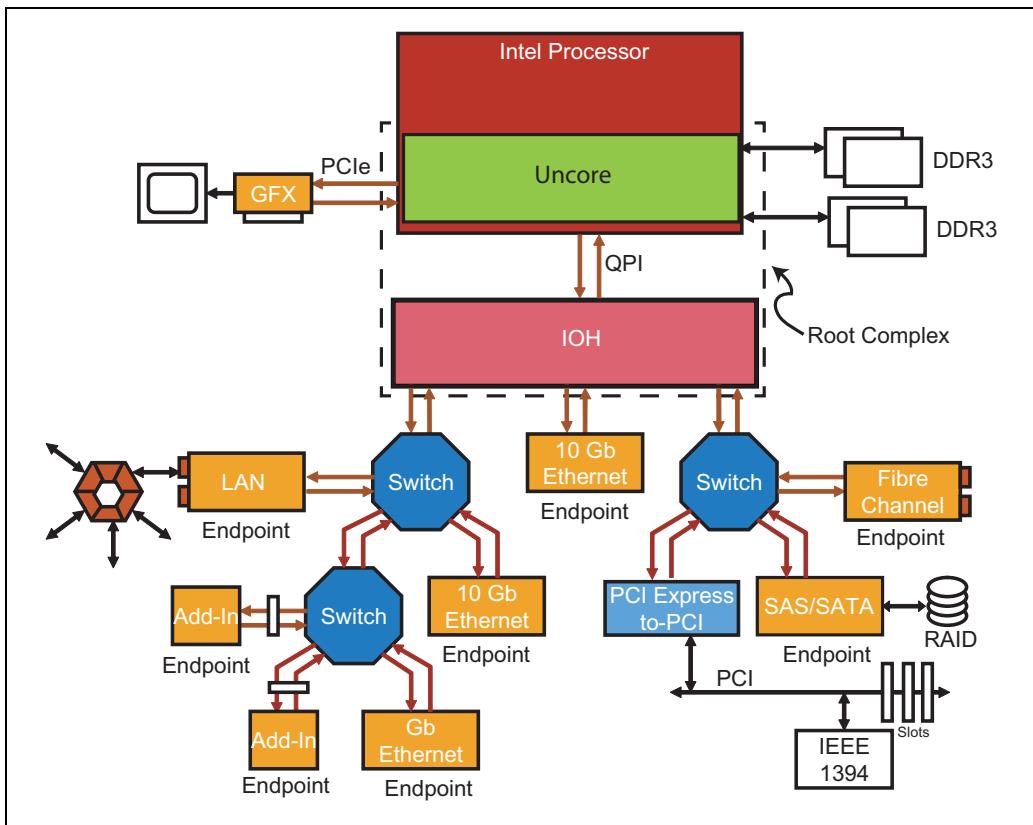
Figure 2-10: Low-Cost PCIe System



# PCI Express Technology

---

Figure 2-11: Server PCIe System



---

## Introduction to Device Layers

PCIe defines a layered architecture as illustrated in Figure 2-12 on page 56. The layers can be considered as being logically split into two parts that operate independently because they each have a transmit side for outbound traffic and a receive side for inbound traffic. The layered approach has some advantages for hardware designers because, if the logic is partitioned carefully, it can be easier to migrate to new versions of the spec by changing one layer of an existing design while leaving the others unaffected. Even so, it's important to note that the layers simply define interface responsibilities and a design is not required to be partitioned according to the layers to be compliant with the spec. The goal in

## Chapter 2: PCIe Architecture Overview

---

this section is to describe the responsibilities of each layer and the flow of events involved in accomplishing a data transfer.

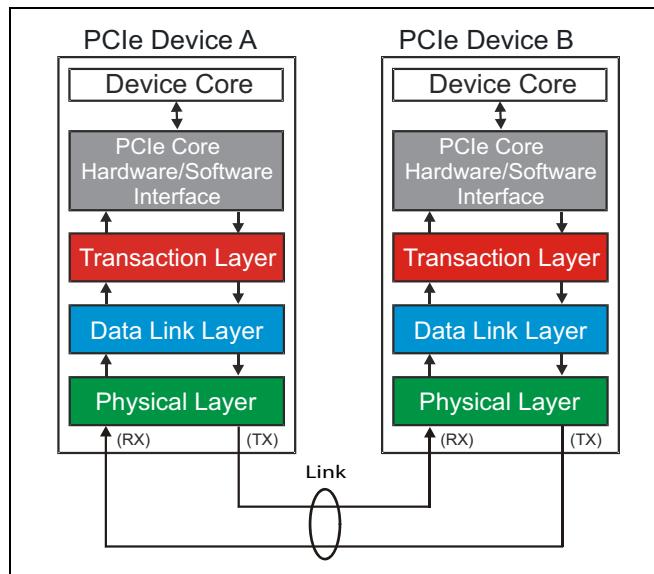
The device layers as shown in Figure 2-12 on page 56 consist of:

- **Device core and interface to Transaction Layer.** The core implements the main functionality of the device. If the device is an endpoint, it may consist of up to 8 functions, each function implementing its own configuration space. If the device is a switch, the switch core consists of packet routing logic and an internal bus for accomplishing this goal. If the device is a root, the root core implements a virtual PCI bus 0 on which resides all the chipset embedded endpoints and virtual bridges.
- **Transaction Layer.** This layer is responsible for Transaction Layer Packet (TLP) creation on the transmit side and TLP decoding on the receive side. This layer is also responsible for Quality of Service functionality, Flow Control functionality and Transaction Ordering functionality. All these four Transaction Layer functions are described in book **Part two**.
- **Data Link Layer.** This layer is responsible for Data Link Layer Packet (DLLP) creation on the transmit side and decoding on the receive side. This layer is also responsible for Link error detection and correction. This Data Link Layer function is referred to as the Ack/Nak protocol. Both these Data Link Layer functions are described in book **Part Three**.
- **Physical Layer.** This layer is responsible for Ordered-Set packet creation on the transmit side and Ordered-Set packet decoding on the receive side. This layer processes all three types of packets (TLPs, DLLPs and Ordered-Sets) to be transmitted on the Link and processes all types of packets received from the Link. Packets are processed on the transmit side by byte striping logic, scramblers, 8b/10b encoders (associated with Gen1/Gen2 protocol) or 128b/130b encoders (associated with Gen3 protocol) and packet serializers. The packet is finally differentially clocking out on all Lanes at the trained Link speed. On the receive Physical Layer, packet processing consists of serially receiving differentially encoded bits and converting to digital format and then deserializing the incoming bit-stream. This is done at a clock rate derived from a recovered clock from the CDR (Clock and Data Recovery) circuit. The received packets are processed by elastic buffers, 8b/10b decoders (associated with Gen1/Gen2 protocol) or 128b/130b decoders (associated with Gen3 protocol), de-scramblers and byte un-striping logic. Finally, the Link Training and Status State Machine (LTSSM) of the Physical Layer is responsible for Link Initialization and Training. All these Physical Layer functions are described in book **Part Four**.

# PCI Express Technology

---

Figure 2-12: PCI Express Device Layers

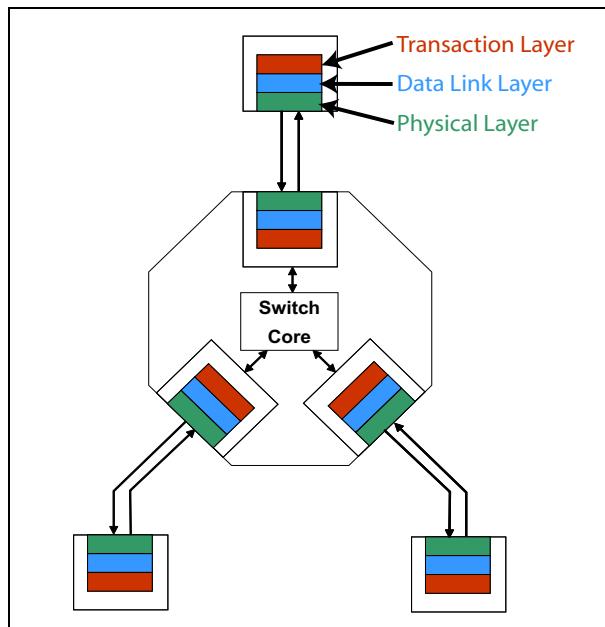


Every PCIe interface supports the functionality of these layers, including Switch Ports, as shown in Figure 2-13 on page 57. A question often came up in earlier classes as to whether a Switch Port needs to implement all the layers, since it's typically only forwarding packets. The answer is yes, and the reason is that evaluating the contents of packets to determine their routing requires looking into the internal details of a packet, and that takes place in the Transaction Layer logic.

## Chapter 2: PCIe Architecture Overview

---

Figure 2-13: Switch Port Layers

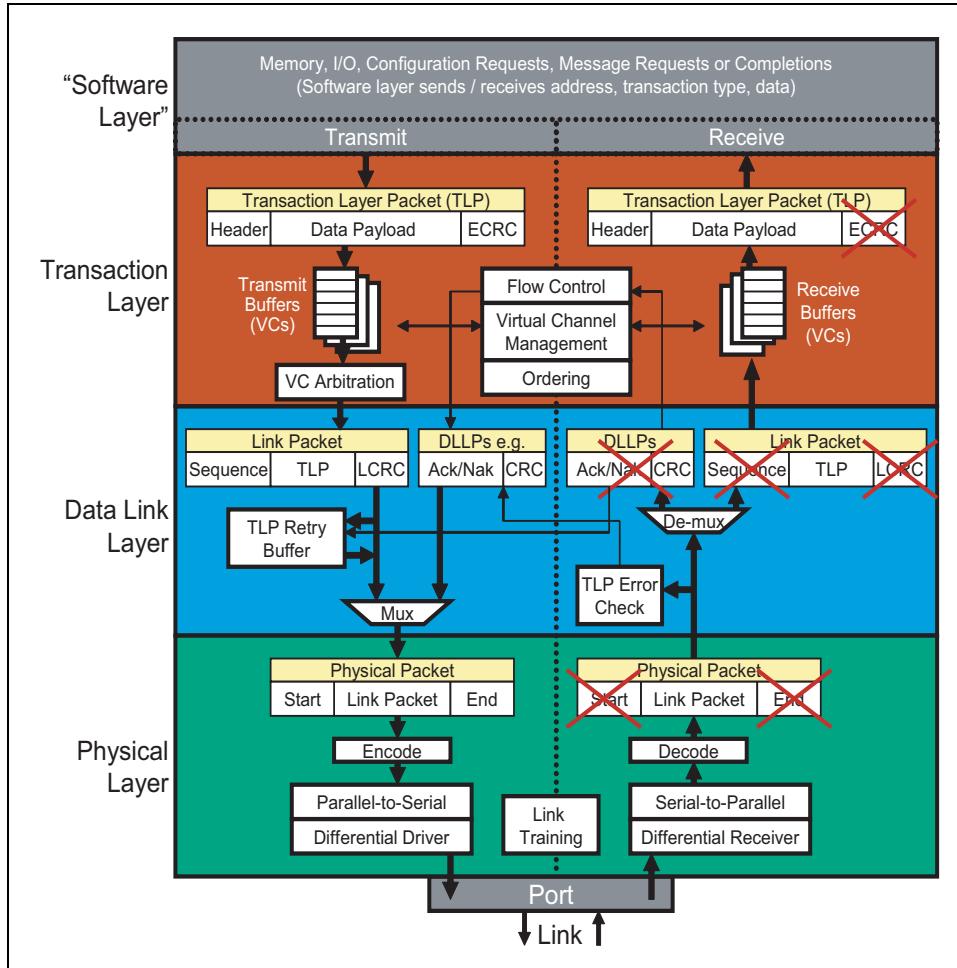


In principle, each layer communicates with the corresponding layer in the device on the other end of the Link. The upper two layers do so by organizing a string of bits into a packet, creating a pattern that is recognizable by the corresponding layer in the receiver. The packets are forwarded through the other layers along the way to get to or from the Link. The Physical Layer also communicates directly with that layer in the other device but it does differently.

Before we go deeper, let's first walk through an overview to see how the layers interact. In broad terms, the contents of an outgoing request or completion packet from the device are assembled in the Transaction Layer based on information presented by the device core logic, which we also sometimes call the Software Layer (although the spec doesn't use that term). That information would usually include the type of command desired, the address of the target device, attributes of the request, and so on. The newly created packet is then stored in a buffer called a Virtual Channel until it's ready for passing to the next layer. When the packet is passed down to the Data Link Layer, additional information is added to the packet for error checking at the neighboring receiver, and a copy is stored locally so we can send it again if a transmission error occurs. When the packet arrives at the Physical Layer it's encoded and transmitted differentially using all the available Lanes of the Link.

# PCI Express Technology

Figure 2-14: Detailed Block Diagram of PCI Express Device's Layers



The receiver decodes the incoming bits in the Physical Layer, checks for errors that can be seen at this level and, if there are none, forwards the resulting packet up to the Data Link Layer. Here the packet is checked for different errors and, if there are no errors, is forwarded up to the Transaction Layer. The packet is buffered, checked for errors, and disassembled into the original information (command, attributes, etc.) so the contents can be delivered to the device core of the receiver. Next, let's explore in greater depth what each of the layers must do to make this process work, using Figure 2-14 on page 58. We start at the top.

# Chapter 2: PCIe Architecture Overview

---

## Device Core / Software Layer

This is the core functionality of the device, such as a network interface or hard drive controller. This isn't defined as a layer in the PCIe spec, but can be thought of in that way since it resides above the Transaction Layer and will be either the source or destination of all Requests. It provides the transmit side of the Transaction Layer with requests that include information like the transaction type, the address, amount of data to transfer, and so on. It's also the destination for information forwarded up from the Transaction Layer when incoming packets have been received.

---

## Transaction Layer

In response to requests from the Software Layer, the Transaction Layer generates outbound packets. It also examines inbound packets and forwards the information contained in them up to the Software Layer. It supports the split transaction protocol for non-posted transactions and associates an inbound Completion with an outbound non-posted Request that was transmitted earlier. The transactions handled by this layer use TLPs (Transaction Layer Packets) and can be grouped into four request categories:

1. Memory
2. IO
3. Configuration
4. Messages

The first three of these were already supported in PCI and PCI-X, but messages are a new type for PCIe. A **Transaction** is defined as the combination of a **Request** packet that a delivers a command to a targeted device, together with any **Completion** packets the target sends back in reply. A list of the request types is given in Table 2-2 on page 59.

*Table 2-2: PCI Express Request Types*

Request Type	Non-Posted or Posted
Memory Read	Non-Posted
Memory Write	Posted
Memory Read Lock	Non-Posted

# PCI Express Technology

---

Table 2-2: PCI Express Request Types (Continued)

Request Type	Non-Posted or Posted
IO Read	Non-Posted
IO Write	Non-Posted
Configuration Read (Type 0 and Type 1)	Non-Posted
Configuration Write (Type 0 and Type 1)	Non-Posted
Message	Posted

The requests also fall into one of two categories as shown in the right column of the table: **non-posted** and **posted**. For non-posted requests, a Requester sends a packet for which a Completer should generate a response in the form of a Completion packet. The reader may recognize this as the split transaction protocol inherited from PCI-X. For example, any read request will be non-posted because the requested data will need to be returned in a completion. Perhaps unexpectedly, IO writes and Configuration writes are also non-posted. Even though they are delivering the data for the command, these requests still expect to receive a completion from the target to confirm that the write data has in fact made it to the destination without error.

In contrast, Memory Writes and Messages are posted, meaning the targeted device does not return a completion TLP to the Requester. Posted transactions improve performance because the Requester doesn't have to wait for a reply or incur the overhead of a completion. The trade-off is that they get no feedback about whether the write has finished or encountered an error. This behavior is inherited from PCI and is still considered a good thing to do because the likelihood of a failure is small and the performance gain is significant. Note that, even though they don't require Completions, Posted Writes do still participate in the Ack/Nak protocol in the Data Link Layer that ensures reliable packet delivery. For more on this, see Chapter 10, entitled "Ack/Nak Protocol," on page 317.

## TLP (Transaction Layer Packet) Basics

A list of all of the PCIe request and completion packet types is given in Table 2-3 on page 61.

## Chapter 2: PCIe Architecture Overview

---

Table 2-3: PCI Express TLP Types

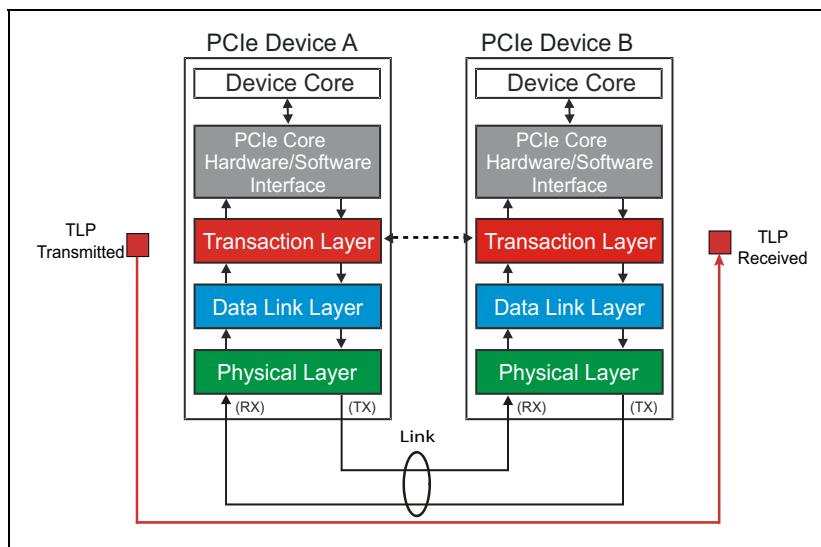
TLP Packet Types	Abbreviated Name
Memory Read Request	MRd
Memory Read Request - Locked access	MRdLk
Memory Write Request	MWr
IO Read	IORD
IO Write	IOWr
Configuration Read (Type 0 and Type 1)	CfgRd0, CfgRd1
Configuration Write (Type 0 and Type 1)	CfgWr0, CfgWr1
Message Request without Data	Msg
Message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

TLPs originate at the Transaction Layer of a transmitter and terminate at the Transaction Layer of a receiver, as shown in Figure 2-15 on page 62. The Data Link Layer and Physical Layer add parts to the packet as it moves through the layers of the transmitter, and then verify at the receiver that those parts were transmitted correctly across the Link.

# PCI Express Technology

---

Figure 2-15: TLP Origin and Destination



**TLP Packet Assembly.** An illustration of the parts of a finished TLP as it is sent over the Link is shown in Figure 2-16 on page 63, where it can be seen that different parts of the packet are added in each of the layers. To make it easier to recognize how the packet gets constructed, the different parts of the TLP are color coded to indicate which layer is responsible for them: red for Transaction Layer, blue for Data Link Layer, and green for the Physical Layer.

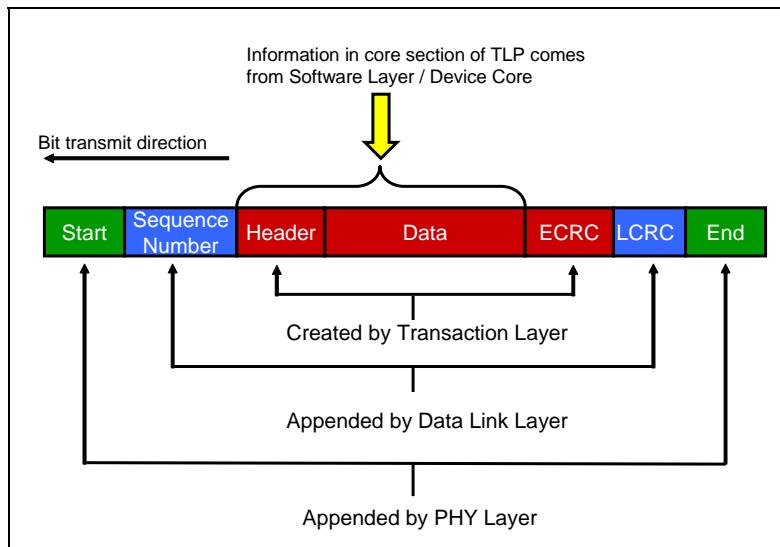
The device core sends the information required to assemble the core section of the TLP in the Transaction Layer. Every TLP will have a header, although some, like a read request, won't contain data. An optional End-to-End CRC (ECRC) field may be calculated and appended to the packet. CRC stands for Cyclic Redundancy Check (or Code) and is employed by almost all serial architectures for the simple reason that it's simple to implement and provides very robust error detection capability. The CRC also detects "burst errors," or string of repeated mistaken bits, up to the length of the CRC value (32 bits for PCIe). Since this type of error is likely to be encountered when sending a long string of bits, this characteristic is very useful for serial transports. The ECRC field is passed unchanged through any service points ("service point" usually refers to a Switch or Root Port that has TLP routing options) between the sender and receiver of the packet, making it useful for verifying at the destination that there were no errors anywhere along the way.

## Chapter 2: PCIe Architecture Overview

For transmission, the core section of the TLP is forwarded to the Data Link Layer, which is responsible to append a Sequence Number and another CRC field called the Link CRC (LCRC). The LCRC is used by the neighboring receiver to check for errors and report the results of that check back to the transmitter for every packet sent on that Link. The thoughtful reader may wonder why the ECRC would be helpful if the mandatory LCRC check already verifies error-free transmission across the Link. The reason is that there is still a place where transmission errors aren't checked, and that is within devices that route packets. A packet arrives and is checked for errors on one port, the routing is checked, and when it's sent out on another port a new LCRC value is calculated and added to it. The internal forwarding between ports could encounter an error that isn't checked as part of the normal PCIe protocol, and that's why ECRC is helpful.

Finally, the resulting packet is forwarded to the Physical Layer where other characters are added to the packet to let the receiver know what to expect. For the first two generations of PCIe, these were control characters added to the beginning and end of the packet. For the third generation, control characters are no longer used but other bits are appended to the blocks that give the needed information about the packets. The packet is then encoded and differentially transmitted on the Link using all of the available lanes.

Figure 2-16: TLP Assembly

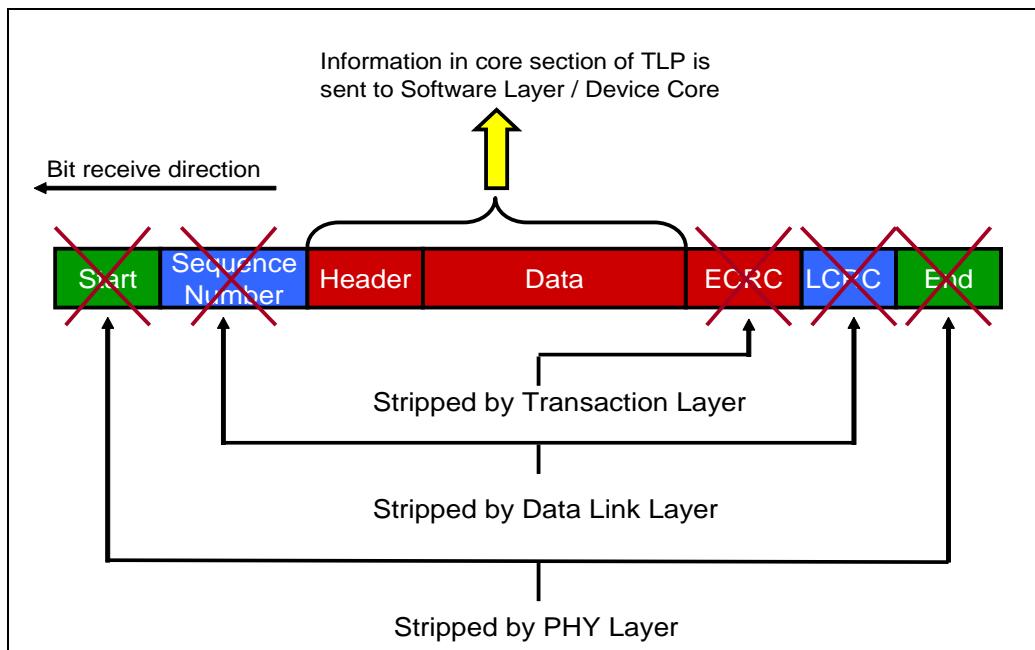


# PCI Express Technology

**TLP Packet Disassembly.** When the neighboring receiver sees the incoming TLP bit stream, it needs to identify and remove the parts that were added to recover the original information requested by the core logic of the transmitter. As shown in Figure 2-17 on page 64, the Physical Layer will verify that the proper Start and End or other characters are present and remove them, forwarding the remainder of the TLP to the Data Link Layer. This layer first checks for LCRC and Sequence Number errors. If no errors are found, it removes those fields from the TLP and forwards it to the Transaction Layer. If the receiver is a Switch, the packet is evaluated in the Transaction Layer to find the routing information in the header of the TLP and determine to which port the packet should be forwarded. Even when it's not the intended destination, a Switch is allowed to check and report an ECRC error if it finds one. However, it's not allowed to modify the ECRC, so the targeted device will be able to detect the ECRC error as well.

The target device can check ECRC errors if it's capable and was enabled. If this is the target device and there was no error, the ECRC field is removed, leaving the header and data portion of the packet to be forwarded to the Software Layer.

Figure 2-17: TLP Disassembly

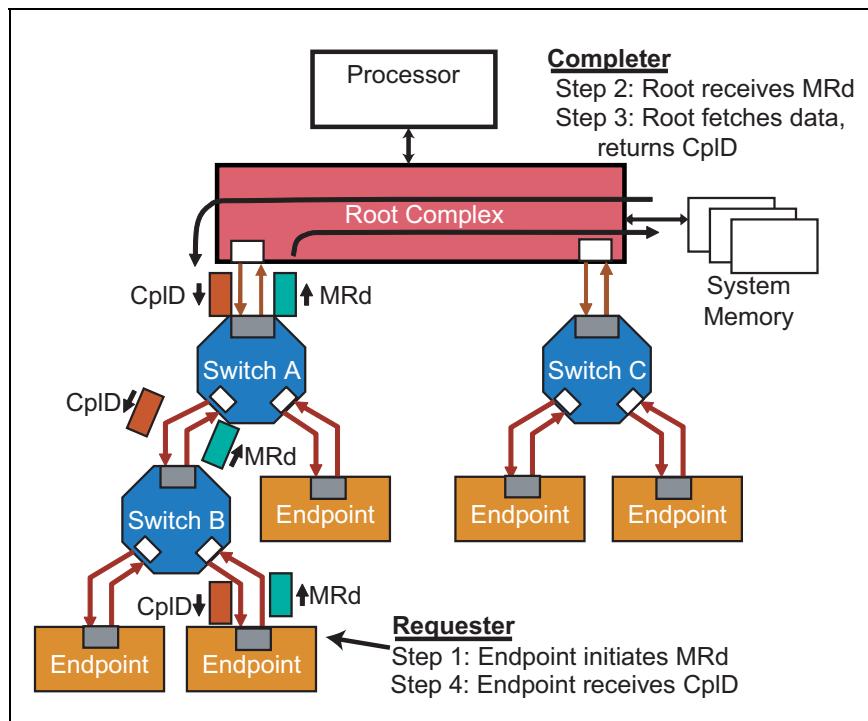


# Chapter 2: PCIe Architecture Overview

## Non-Posted Transactions

**Ordinary Reads.** Figure 2-18 on page 65 shows an example of a Memory Read Request sent from an Endpoint to system memory. A detailed discussion of the TLP contents can be found in Chapter 5, entitled "TLP Elements," on page 169, but an important part of any memory read request is the target address. The address for a memory Request can be 32 or 64 bits, and determines the packet routing. In this example, the request gets routed through two Switches that forward it up to the target, which is the Root in this case. When the Root decodes the request and recognizes that the address in the packet targets system memory, it fetches the requested data. To return that data to the Requester, the Transaction Layer of the Root Port creates as many Completions as are needed to deliver all the requested data to the Requester. The largest possible data payload for PCIe is 4 KB per packet, but devices are often designed to use smaller payloads than that, so several completions may be needed to return a large amount of data.

Figure 2-18: Non-Posted Read Example



# PCI Express Technology

---

Those Completion packets also contain routing information to direct them back to the Requester, and the Requester includes its return address for this purpose in the original request. This “return address” is simply the Device ID of the Requester as it was defined for PCI, which is a combination of three things: its PCI Bus number in the system, its Device number on that bus, and its Function number within that device. This Bus, Device, and Function number information (sometimes abbreviated as BDF) is the routing information that Completions will use to get back to the original Requester. As was true for PCI-X, a Requester can have several split transactions in progress at the same time and must be able to associate incoming completions with the correct requests. To facilitate that, another value was added to the original request called a Tag that is unique to each request. The Completer copies this transaction Tag and uses it in the Completion so the Requester can quickly identify which Request this Completion is servicing.

Finally, a Completer can also indicate error conditions by setting bits in the completion status field. That gives the Requester at least a broad idea of what might have gone wrong. How the Requester handles most of these errors will be determined by software and is outside the scope of the PCIe spec.

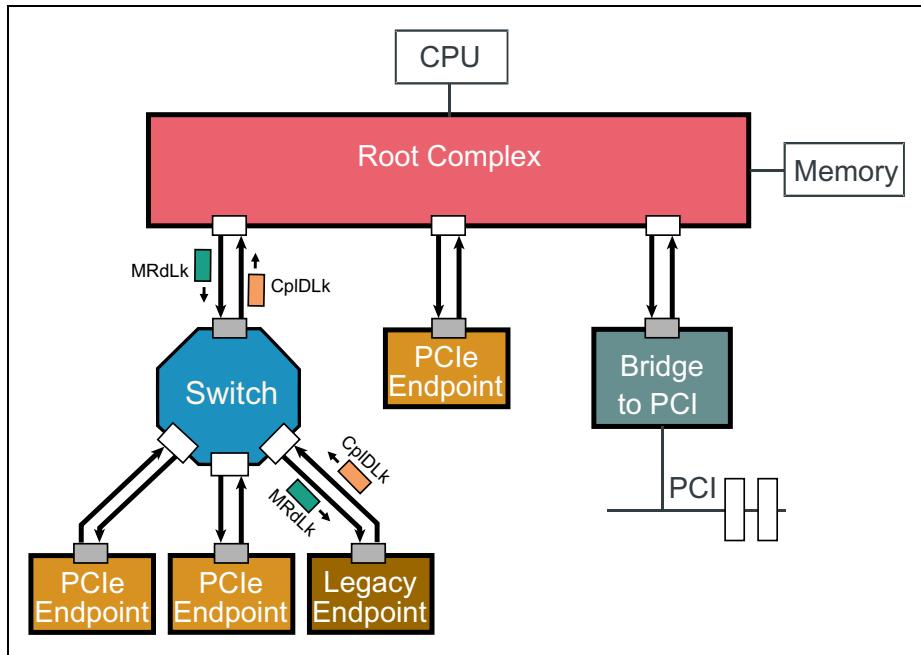
**Locked Reads.** Locked Memory Reads are intended to support what are called Atomic Read-Modify-Write operations, a type of uninterruptable transaction that processors use for tasks like testing and setting a semaphore. While the test and set is in progress, no other access to the semaphore can take place or a race condition could develop. To prevent this, processors use a lock indicator (such as a separate pin on the parallel Front-Side Bus) that prevents other transactions on the bus until the locked one is finished. What follows here is just a high level introduction to the topic. For more information on Locked transactions, refer to Appendix D called “Appendix D: Locked Transactions” on page 963.

As a bit of history, in the early days of PCI the spec writers anticipated cases where PCI would actually replace the processor bus. Consequently, support for things that a processor would need to do on the bus were included in the PCI spec, such as locked transactions. However, PCI was only rarely ever used this way and, in the end, much of this processor bus support was dropped. Locked cycles remained, though, to support a few special cases, and PCIe carries this mechanism forward for legacy support. Perhaps to speed migration away from its use, new PCIe devices are prohibited from accepting locked requests; it’s only legal for those that self-identify as Legacy Devices. In the example shown in Figure 2-19 on page 67, a Requester begins the process by sending a locked request (MRdLk). By definition, such a request is only allowed to come from the CPU, so in PCIe only a Root Port will ever initiate one of these.

## Chapter 2: PCIe Architecture Overview

The locked request is routed through the topology using the target memory address and eventually reaches the Legacy Endpoint. As the packet makes its way through each routing device (called a service point) along the way, the Egress Port for the packet is locked, meaning no other packets will be allowed in that direction until the path is unlocked.

Figure 2-19: Non-Posted Locked Read Transaction Protocol



When the Completer receives the packet and decodes its contents, it gathers the data and creates one or more Locked Completions with data. These Completions are routed back to the Requester using the Requester ID, and each Egress Port they pass through is then locked, too.

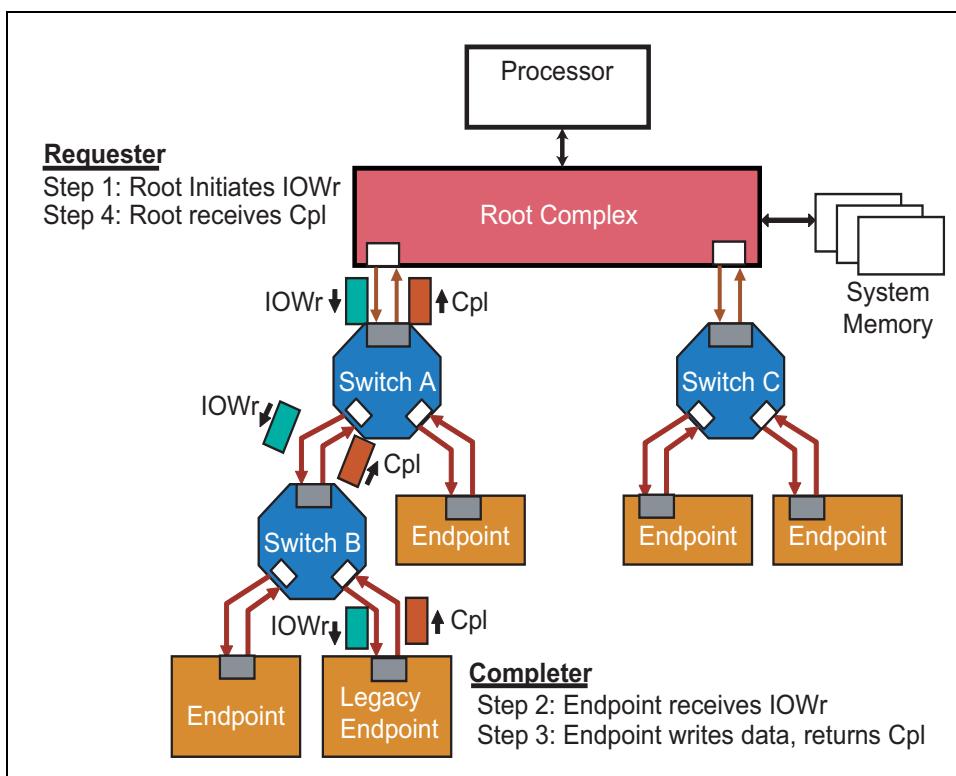
If the Completer encounters a problem, it returns a locked completion packet without data (the original read should have resulted in data so if there isn't any we know there's been a problem) and the status field will indicate something about the error. The Requester will understand that to mean that the lock did not succeed and so the transaction will be cancelled and software will need to decide what to do next.

# PCI Express Technology

**IO and Configuration Writes.** Figure 2-20 on page 68 illustrates a non-posted IO write transaction. Like a locked request, an IO cycle can also legally target only a Legacy Endpoint. The request is routed through the Switches based on the IO address until it reaches the target Endpoint. When the Completer receives the request, it accepts the data and returns a single completion packet without data that confirms reception of the packet. The status field in the completion would report whether an error had occurred and, if so, the Requester's software would handle it.

If the completion reports no errors the Requester knows that the write data has been successfully delivered and the next step in the sequence of instructions for that Completer is now permitted. And that really summarizes the motivation for the non-posted write: unlike a memory write, it's not enough to know that the data **will** get to the destination sometime in the future. Instead, the next step can't logically take place until we know that it **has** gotten there. As with locked cycles, non-posted writes can only come from the processor.

Figure 2-20: Non-Posted Write Transaction Protocol

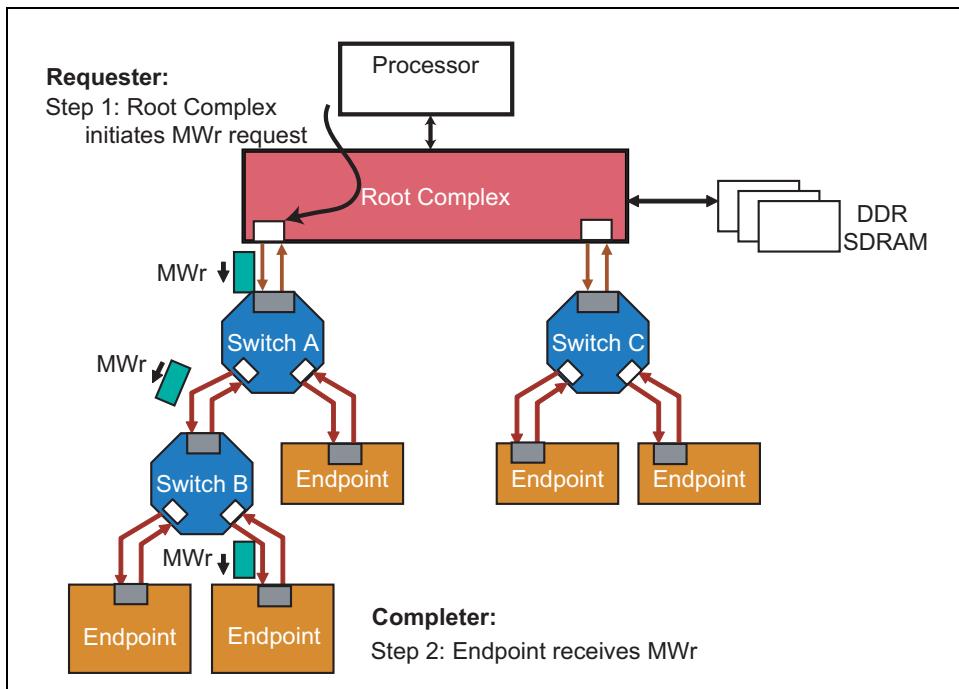


# Chapter 2: PCIe Architecture Overview

## Posted Writes

**Memory Writes.** Memory writes are always posted and never receive completions. Once the request has been sent, the Requester doesn't wait for any feedback before going on to the next request, and no time or bandwidth is spent returning a completion. As a result, posted writes are faster and more efficient than non-posted requests and improve system performance. As shown in Figure 2-21 on page 69, the packet is routed through the system using its target memory address to the Completer. Once a Link has successfully sent the request, that transaction is finished on that Link and its available for other packets. Eventually, the Completer accepts the data and the transaction is truly finished. Of course, one trade-off with this approach is that, since no Completion packets are sent, there's also no means for reporting errors back to the Requester. If the Completer encounters an error, it can log it and send a Message to the Root to inform system software about the error, but the Requester won't see it.

Figure 2-21: Posted Memory Write Transaction Protocol



# PCI Express Technology

---

**Message Writes.** Interestingly, unlike the other requests we've looked at so far, there are several possible routing methods for messages, and a field within the message indicates which type to use. For example, some messages are posted write requests that target a specific Completer, others are broadcast from the Root to all Endpoints, while still others sent from an Endpoint are automatically routed to the Root. To learn more about the different types of routing refer to Chapter 4, entitled "Address Space & Transaction Routing," on page 121.

Messages are useful in PCIe to help achieve a design goal of lowering the pin count. They eliminate the need for the side-band signals that PCI used to report things like interrupts, power management events, and errors because they can report that information in a packet over the normal data path.

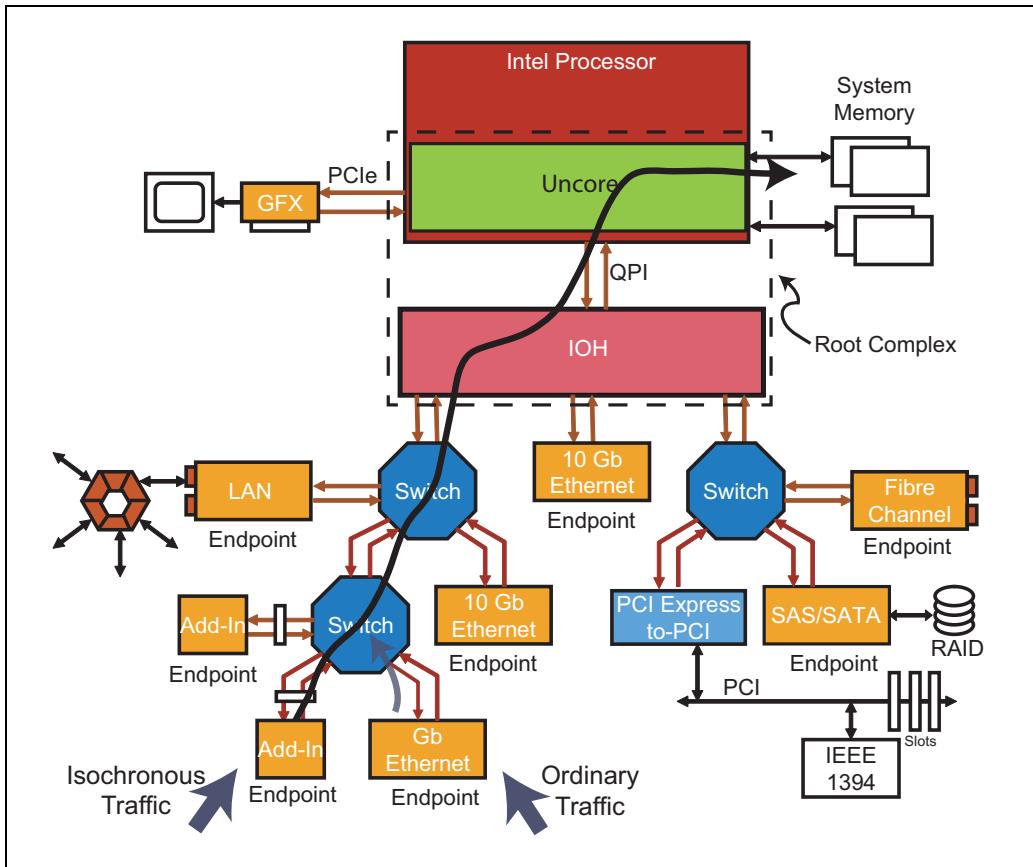
## Quality of Service (QoS)

PCIe was designed from its inception to be able to support time-sensitive transactions for applications like streaming audio or video where data delivery must be timely in order to be useful. This is referred to as providing Quality of Service and is accomplished by the addition of a few things. First, each packet is assigned a priority by software by setting a 3-bit field within it called Traffic Class (TC). Generally speaking, assigning a higher-numbered TC to a packet is expected to give it a higher priority in the system. Second, multiple buffers, called Virtual Channels (VC), are built into the hardware for each port and a packet is placed into the appropriate buffer based on its TC. Third, since a port now has multiple buffers with packets available for transmission at a given time, arbitration logic is needed to select among the VCs. Finally, Switches must select between competing input ports for access to the VCs of a given output port. This is called Port Arbitration and can be hardware assigned or software programmable. All of these hardware pieces must be in place to allow a system to prioritize packets. If properly programmed and set up, such a system can even provide guaranteed service for a given path.

To illustrate the concept, consider Figure 2-22 on page 71, in which a video camera and SCSI device both need to send data to system DRAM. The difference is that the camera data is time critical; if the transmission path to the target device is unable to keep up with its bandwidth, frames will get dropped. The system needs to be able to guarantee a bandwidth that's at least as high as the camera or the captured video may appear choppy. At the same time, the SCSI data needs to be delivered without errors, but how long it takes is not as important. Clearly, then, when both a video data packet and a SCSI packet need to be sent at the same time, the video traffic should have a higher priority. QoS refers to the ability of the system to assign different priorities to packets and route them through the topology with deterministic latencies and bandwidth. For more detail on QoS, refer to Chapter 7, entitled "Quality of Service," on page 245.

## Chapter 2: PCIe Architecture Overview

Figure 2-22: QoS Example



### Transaction Ordering

Within a VC, the packets normally all flow through in the same order in which they arrived, but there are exceptions to this general rule. PCI Express protocol inherits the PCI transaction-ordering model, including support for relaxed-ordering cases added with the PCI-X architecture. These ordering rules guarantee that packets using the same traffic class will be routed through the topology in the correct order, preventing potential deadlock or live-lock conditions. An interesting point to note is that, since ordering rules only apply within a VC and packets that use different TCs may not get mapped into the same VC, packets using different TCs are understood by software to have no ordering relationship. This ordering is maintained in the VCs within the transaction layer.

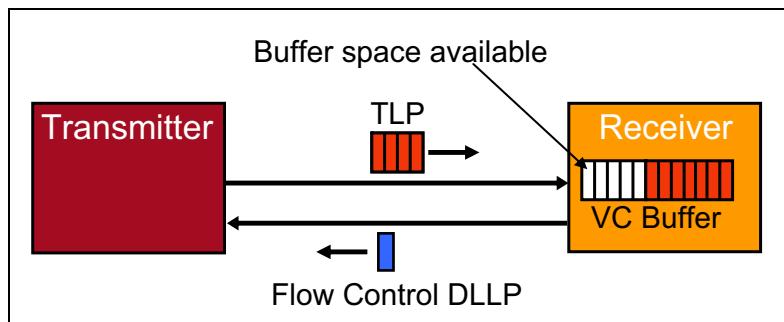
# PCI Express Technology

---

## Flow Control

A typical protocol used by serial transports is to require that a transmitter only send a packet to its neighbor if there is sufficient buffer space to receive it. That cuts down on performance-wasting events on the bus like the disconnects and retries that PCI allowed and thus removes that class of problems from the transport. The trade-off is that the receiver must report its buffer space often enough to avoid unnecessary stalls and that reporting takes a little bandwidth of its own. In PCIe this reporting is done with DLLPs (Data Link Layer Packets), as we'll see in the next section. The reason is to avoid a possible deadlock condition that might occur if TLPs were used, in which a transmitter can't get a buffer size update because its own receive buffer is full. DLLPs can always be sent and received regardless of the buffer situation, so that problem is avoided. This flow control protocol is automatically managed at the hardware level and is transparent to software.

Figure 2-23: Flow Control Basics



As shown in Figure 2-23 on page 72, the Receiver contains the VC Buffers that hold received TLPs. The Receiver advertises the size of those buffers to the Transmitters using Flow Control DLLPs. The Transmitter tracks the available space in the Receiver's VC Buffers and is not allowed to send more packets than the Receiver can hold. As the Receiver processes the TLPs and removes them from the buffer, it periodically sends Flow Control Update DLLPs to keep the Transmitter up-to-date regarding the available space. To learn more about this, see Chapter 6, entitled "Flow Control," on page 215.

---

## Data Link Layer

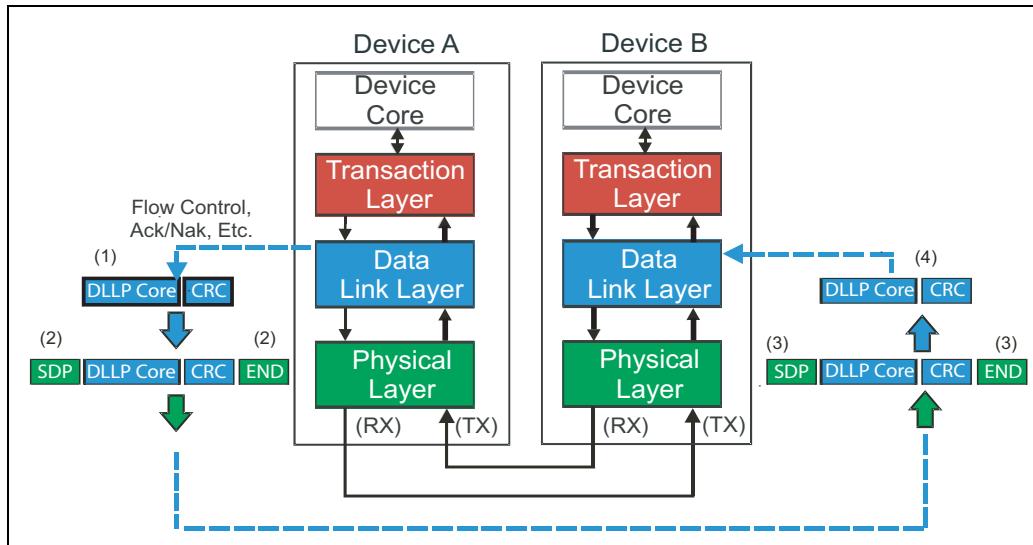
This logic is responsible for Link management and performs three major functions: TLP error correction, flow control, and some Link power management. It accomplishes these by generating DLLPs as shown in Figure 2-24 on page 73.

# Chapter 2: PCIe Architecture Overview

## DLLPs (Data Link Layer Packets)

DLLPs are transferred between Data Link Layers of the two neighboring devices on a Link. The Transaction Layer is not even aware of these packets, which only travel between neighboring devices and are not routed anywhere else. They are small (always just 8 bytes) compared to TLPs, and that's a good thing because they represent overhead for maintaining Link protocol.

Figure 2-24: DLLP Origin and Destination



**DLLP Assembly.** As shown in Figure 2-24 on page 73, a DLLP originates at the Data Link Layer of the transmitter and is consumed by the Data Link Layer of the receiver. A 16-bit CRC is added to the DLLP Core to check for errors at the receiver. The DLLP contents are forwarded to the Physical Layer which appends a Start and End character to the packet (for the first two generations of PCIe), and then encodes and differentially transmits it over the Link using all the available lanes.

**DLLP Disassembly.** When a DLLP is received by the Physical Layer, the bit stream is decoded and the Start and End frame characters are removed. The rest of the packet is forwarded to the Data Link Layer, which checks for CRC errors and then takes the appropriate action based on the packet. The Data Link Layer is the destination for the DLLP, so it isn't forwarded up to the Transaction Layer.

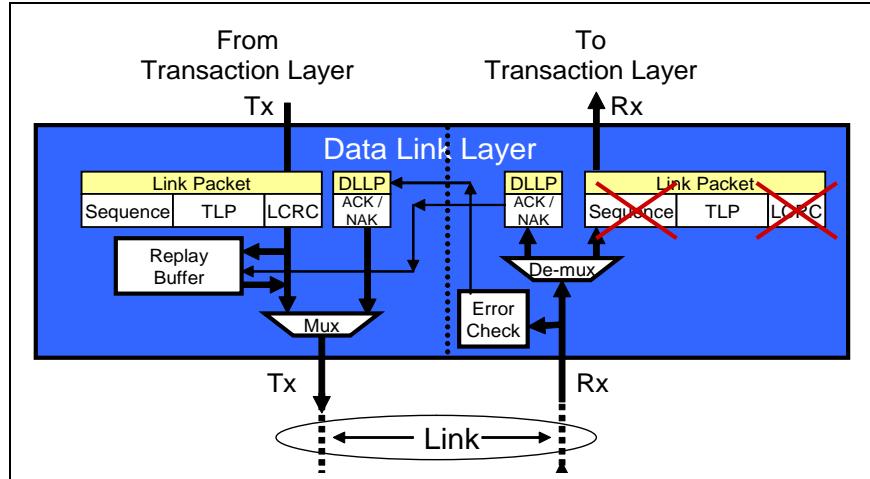
# PCI Express Technology

## Ack/Nak Protocol

The error correction function, illustrated in Figure 2-25 on page 74, is provided through a hardware-based automatic retry mechanism. As shown in Figure 2-26 on page 75, an LCRC and Sequence Number are added to each outgoing TLP and checked at the receiver. The transmitter's Replay Buffer holds a copy of every TLP that has been sent until receipt at the neighboring device has been confirmed. That confirmation takes the form of an Ack DLLP (positive acknowledgement) sent by the Receiver with the Sequence Number of the last good TLP it has seen. When the Transmitter sees the Ack, it flushes the TLP with that Sequence Number out of the Replay Buffer, along with all the TLPs that were sent before the one that was acknowledged.

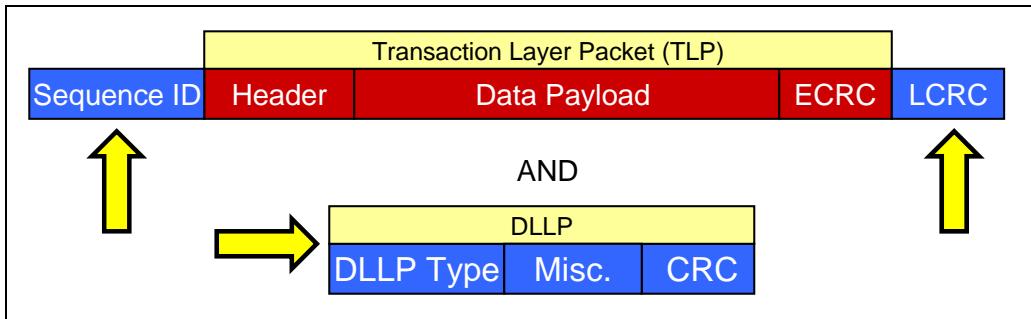
If the Receiver detects a TLP error, it drops the TLP and returns a Nak to the Transmitter, which then replays all unacknowledged TLPs in hopes of a better result the next time. Since detected errors are almost always transient events, a replay will very often correct the problem. This process is often referred to as the Ack/Nak protocol.

Figure 2-25: Data Link Layer Replay Mechanism



## Chapter 2: PCIe Architecture Overview

Figure 2-26: TLP and DLLP Structure at the Data Link Layer



The basic form of a DLLP is also shown in Figure 2-26 on page 75, and consists of a 4-byte DLLP type field that may include some other information and a 2-byte CRC.

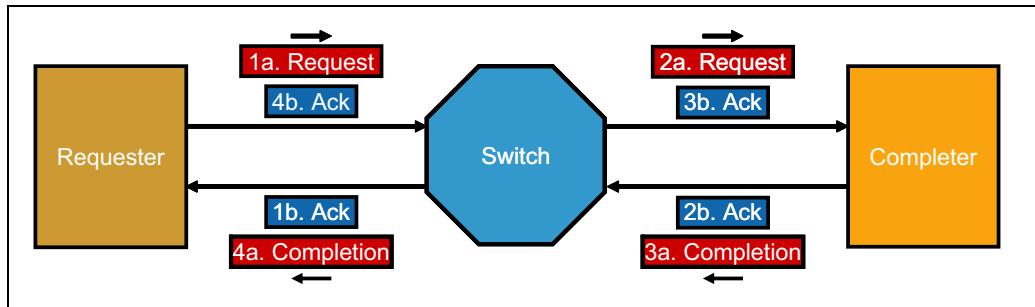
Figure 2-27 on page 76 shows an example of a memory read going across a Switch. In general, the steps for this case would be as follows:

1. **Step 1a:** Requester sends a memory read request and saves a copy in its Replay Buffer. Switch receives the MRd TLP and checks the LCRC and Sequence Number.  
**Step 1b:** No error is seen, so the Switch returns an Ack DLLP to Requester. In response, Requester discards its copy of the TLP from the Replay Buffer.
2. **Step 2a:** Switch forwards the MRd TLP to the correct Egress Port using memory address for its routing and saves a copy in the Egress Port's Replay Buffer. The Completer receives the MRd TLP and checks for errors.  
**Step 2b:** No error is seen, so the Completer returns an Ack DLLP to the Switch. Switch Port purges its copy of the MRd TLP from its Replay Buffer.
3. **Step 3a:** As the final destination of the request, the Completer checks the optional ECRC field in MRd TLP. No errors are seen so the request is passed to the core logic. Based on the command, the device fetches the requested data and returns a Completion with Data TLP (CplD) while saving a copy in its Replay Buffer. Switch receives CplD TLP and checks for errors.  
**Step 3b:** No error is seen, so the Switch returns an Ack DLLP to the Completer. Completer discards its copy of the CplD TLP from its Replay Buffer.
4. **Step 4a:** Switch decodes the Requester ID field in CplD TLP and routes the packet to the correct Egress Port, saving a copy in the Egress Port's Replay Buffer. Requester receives CplD TLP and checks for errors.  
**Step 4b:** No error is seen, so the Requester returns Ack DLLP to Switch. Switch discards its copy of the CplD TLP from its Replay Buffer. Requester checks the optional ECRC field and finds no error, so data is passed up to the core logic.

# PCI Express Technology

---

Figure 2-27: Non-Posted Transaction with Ack/Nak Protocol



## Flow Control

The second major Link Layer function is Flow Control. Following power-up or Reset, this mechanism is initialized by the Data Link Layer automatically in hardware and then updated during run-time. An overview of this was already presented in the section on TLPs so that won't be repeated here. To learn more about this topic, see Chapter 6, entitled "Flow Control," on page 215.

## Power Management

Finally, the Link Layer participates in power management, as well, because DLLPs are used to communicate the requests and handshakes associated with Link and system power states. For a detailed discussion on this topic, refer to Chapter 16, entitled "Power Management," on page 703.

---

## Physical Layer

### General

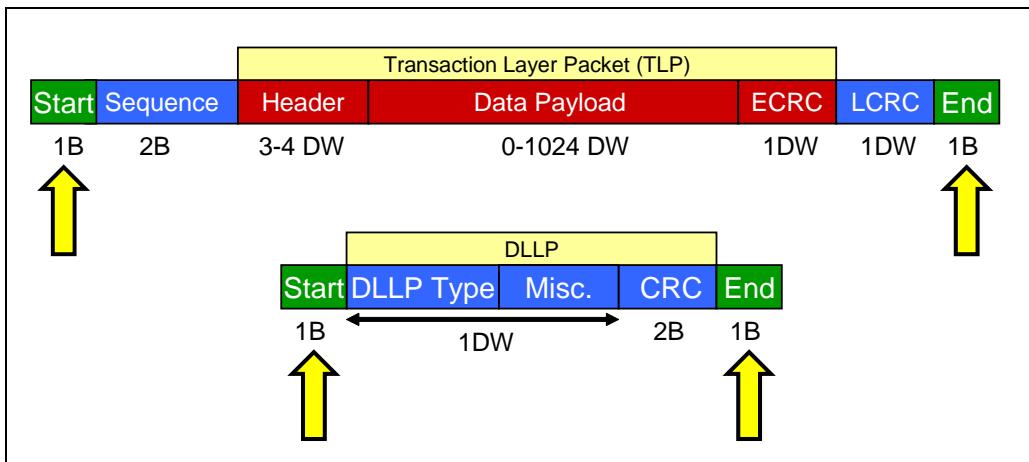
The Physical Layer is the lowest hierarchical layer for PCIe as shown in Figure 2-14 on page 58. Both TLP and DLLP type packets are forwarded down from the Data Link Layer to the Physical Layer for transmission over the Link and forwarded up to the Data Link Layer at the Receiver. The spec divides the Physical Layer discussion into two portions: a logical part and an electrical part, and we'll preserve that split here as well. The Logical Physical Layer contains the digital logic associated with preparing the packets for serial transmission on the Link and reversing that process for inbound packets. The Electrical Physical Layer is the analog interface of the Physical Layer that connects to the Link and consists of differential drivers and receivers for each lane.

## Chapter 2: PCIe Architecture Overview

### Physical Layer - Logical

TLPs and DLLPs from the Data Link Layer are clocked into a buffer in the Physical Layer, where Start and End characters are added to facilitate detection of the packet boundaries at the receiver. Since the Start and End characters appear on both ends of a packet they are also called “framing” characters. The framing characters are shown appended to a TLP and DLLP in Figure 2-28 on page 77, which also shows the size of each field.

Figure 2-28: TLP and DLLP Structure at the Physical Layer



Within this layer, each byte of a packet is split out across all of the lanes in use for the Link in a process called byte striping. Effectively, each lane operates as an independent serial path across the Link and their data is all aggregated back together at the receiver. Each byte is scrambled to reduce repetitive patterns on the transmission line and reduce EMI (electro-magnetic interference) seen on the Link. For the first two generations of PCIe (Gen1 and Gen2 PCIe), the 8-bit characters are encoded into 10-bit “symbols” using what is called 8b/10b encoding logic. This encoding adds overhead to the outgoing data stream, but also adds a number of useful characteristics (for more on this, see “8b/10b Encoding” on page 380). Gen3 Physical Layer logic when transmitting at Gen3 speed, does not encode the packet bytes using 8b/10b encoding. Rather another encoding scheme referred to as 128b/130b encoding is employed with the packet bytes scrambled transmitted. The 10b symbols on each Lane (Gen1 and Gen2) or the packet bytes on each Lane (Gen3) are then serialized and clocked out differentially on each Lane of the Link at 2.5 GT/s (Gen1), or 5 GT/s (Gen2) or 8 GT/s (Gen3).

# PCI Express Technology

---

Receivers clock in the packet bits at the trained clock speeds as they arrive on all lanes. If 8b/10b is in use (at Gen1 and Gen2 mode), the serial bit stream of the packet is converted into 10-bit symbols using a deserializer so it's ready for 8b/10b decoding. However, before decoding, the symbols pass through an elastic buffer, a clever device that compensates for the slight difference in frequency between the internal clocks of two connected devices. Next, the 10-bit symbol stream is decoded back to the proper 8-bit characters via an 8b/10b decoder. Gen3 Physical Layer logic, when receiving serial bit stream of the packet at Gen3 speed, will convert it into a byte stream using a deserializer that has established block lock. The byte stream is passed through an elastic buffer which does clock tolerance compensation. The 8b/10b decoder stage is skipped given packets clocked at Gen3 speeds are not 8b/10b encoded. The 8-bit characters on all lanes are de-scrambled, the bytes from all the lanes are un-striped back into a single character stream and, finally, the original data stream from the Transmitter is recovered.

## Link Training and Initialization

Another responsibility of the Physical Layer is the initialization and training process on the Link. In this fully automatic process, several steps are taken to prepare the Link for normal operation, which involves determining the status of several optional conditions. For example, the Link width can be from one lane to 32 lanes, and multiple speeds might be available. The training process will discover these options and go through a state machine sequence to resolve the best combination. In that process, several things are checked or established to ensure proper and optimal operation, such as:

- Link width
- Link data rate
- Lane reversal - Lanes connected in reverse order
- Polarity inversion - Lane polarity connected backward
- Bit lock per Lane - Recovering the transmitter clock
- Symbol lock per Lane - Finding a recognizable position in the bit-stream
- Lane-to-Lane de-skew within a multi-Lane Link.

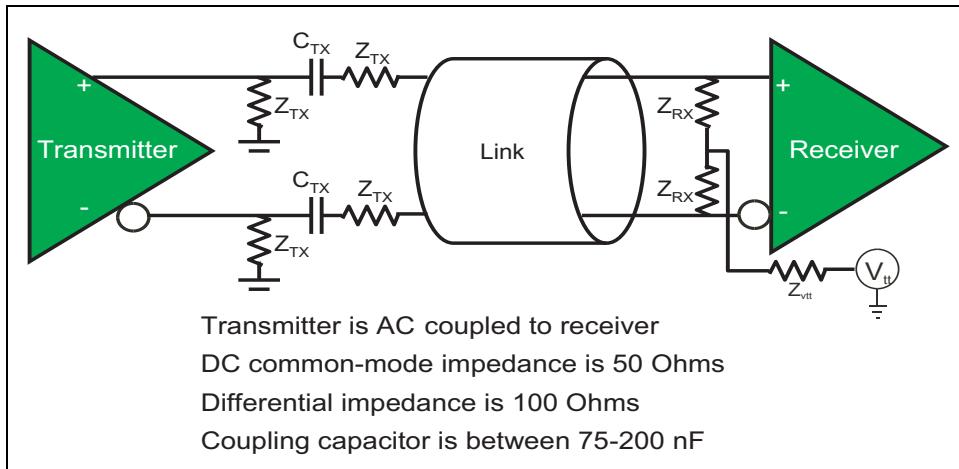
## Physical Layer - Electrical

The physical sender and receiver on a Link are connected with an AC-coupled Link as shown in Figure 2-29 on page 79. The term "AC-coupled" simply means that a capacitor resides physically in the path between the devices and serves to pass the high-frequency (AC) component of the signal while blocking the low-frequency (DC) part. Many serial transports use this approach because it allows the common mode voltage (the level at which the positive and negative versions

## Chapter 2: PCIe Architecture Overview

of the signal cross) to be different at the transmitter and receiver, meaning they're not required to have the same reference voltage. This isn't a big issue if the two devices are nearby and in the same box, but if they were in different buildings it would be very difficult for them to have a common reference voltage that was precisely the same.

Figure 2-29: Physical Layer Electrical



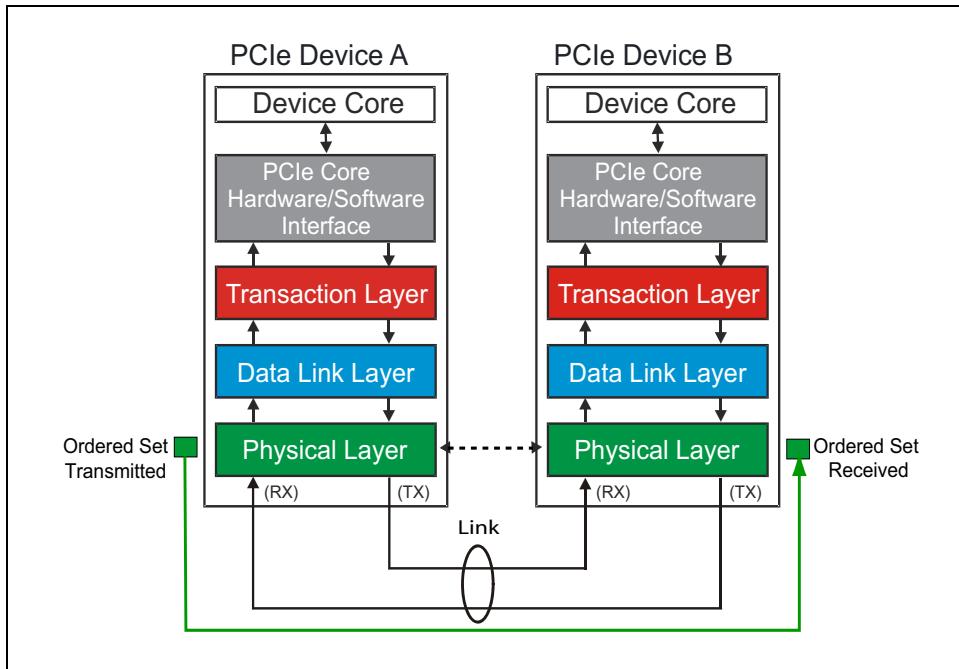
### Ordered Sets

The last type of traffic sent between devices uses only the Physical Layers. Although easily recognized by the receiver, this information is not technically in the form of a packet because it doesn't have Start and End characters, for example. Instead, it's organized into what are called Ordered Sets that originate at the Transmitter's Physical Layer terminate at the Receiver's Physical Layer, as shown in Figure 2-30 on page 80. For Gen1 and Gen2 data rates, an Ordered Set starts with a single COM character followed by three or more other characters that define the information to be sent. The nomenclature for the type of characters used in PCIe is discussed in more detail in "Character Notation" on page 382; for now it's enough to say that the COM character has characteristics that make it work well for this purpose. Ordered Sets are always a multiple of 4 bytes in size, and an example is shown in Figure 2-31 on page 80. In Gen3 mode of operation, the Ordered Set format is different from Gen1/Gen2 described above. Details to be covered in Chapter 14, entitled "Link Initialization & Training," on page 505. Ordered Sets always terminate at the neighboring device and are not routed through the PCIe fabric.

# PCI Express Technology

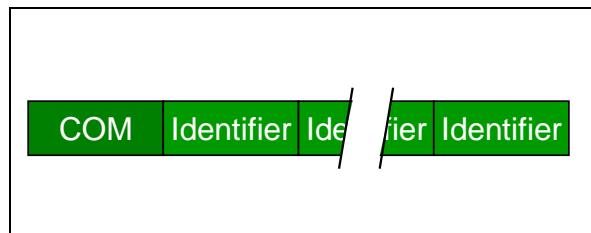
---

Figure 2-30: Ordered Sets Origin and Destination



Ordered Sets are used in the Link Training process, as described in Chapter 14, entitled "Link Initialization & Training," on page 505. They're also used to compensate for the slight differences between the internal clocks of the transmitter and receiver, a process called clock tolerance compensation. Finally, Ordered Sets are used to indicate entry into or exit from a low power state on the Link.

Figure 2-31: Ordered-Set Structure



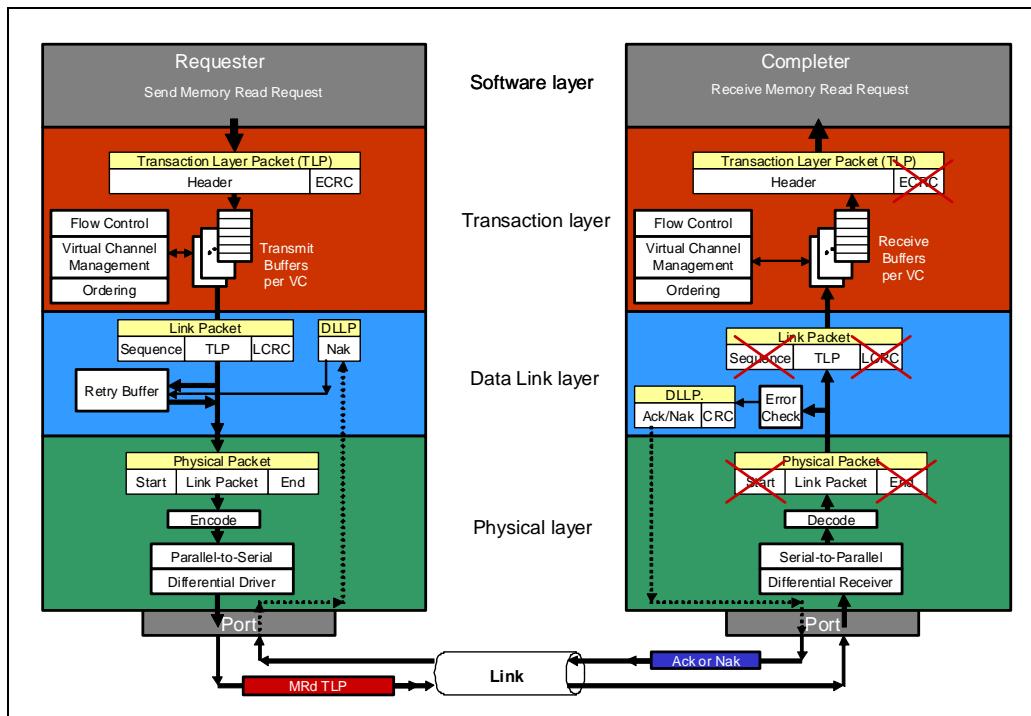
## Protocol Review Example

At this point, let's review the overall Link protocol by using an example to illustrate the steps that take place from the time a Requester initiates a memory read request until it obtains the requested data from a Completer.

### Memory Read Request

For the first part of the discussion, refer to Figure 2-32 on page 81. The Requester's Device Core or Software Layer sends a request to the Transaction Layer and includes the following information: 32-bit or 64-bit memory address, transaction type, amount of data to read calculated in dwords, traffic class, byte enables, attributes etc.

Figure 2-32: Memory Read Request Phase



# PCI Express Technology

---

The Transaction layer uses this information to build a MRd TLP. The details of the TLP packet format are described later, but for now it's enough to say that a 3 DW or 4 DW header is created depending on address size (32-bit or 64-bit). In addition, the Transaction Layer adds the Requester ID (bus#, device#, function#) to the header so the Completer can use that to return the completion. The TLP is placed in the appropriate virtual channel buffer to wait its turn for transmission. Once the TLP has been selected, the Flow Control logic confirms there is sufficient space available in the neighboring device's receive buffer (VC), and then the memory read request TLP is sent to the Data Link Layer.

The Data Link Layer adds a 12-bit Sequence Number and a 32-bit LCRC value to the packet. A copy of the TLP with Sequence Number and LCRC is stored in the Replay Buffer and the packet is forwarded to the Physical Layer.

In the Physical Layer the Start and End characters are added to the packet, which is then byte striped across the available Lanes, scrambled, and 8b/10b encoded. Finally the bits are serialized on each lane and transmitted differentially across the Link to the neighbor.

The Completer de-serializes the incoming bit stream back into 10-bit symbols and passes them through the elastic buffer. The 10-bit symbols are decoded back to bytes and the bytes from all Lanes are de-scrambled and un-striped. The Start and End characters are detected and removed. The rest of the TLP is forwarded up to the Data Link Layer.

The Completer's Data Link Layer checks for LCRC errors in the received TLP and checks the Sequence Number for missing or out-of-sequence TLPs. If there's no error, it creates an Ack that contains the same Sequence Number that was used in the read request. A 16-bit CRC is calculated and appended to the Ack contents to create a DLLP that is sent back to the Physical Layer which adds the proper framing symbols and transmits the Ack DLLP to the Requester.

The Requester Physical Layer receives the Ack DLLP, checks and removes the framing symbols, and forwards it up to the Data Link Layer. If the CRC is valid, it compares the acknowledged Sequence Number with the Sequence Numbers of the TLPs stored in the Replay Buffer. The stored memory read request TLP associated with the Ack received is recognized and that TLP is discarded from the Replay Buffer. If a Nak DLLP was received by the Requester instead, it would re-send a copy of the stored memory read request TLP. Since the DLLP only has meaning to the Data Link Layer, nothing is forwarded to the Transaction Layer.

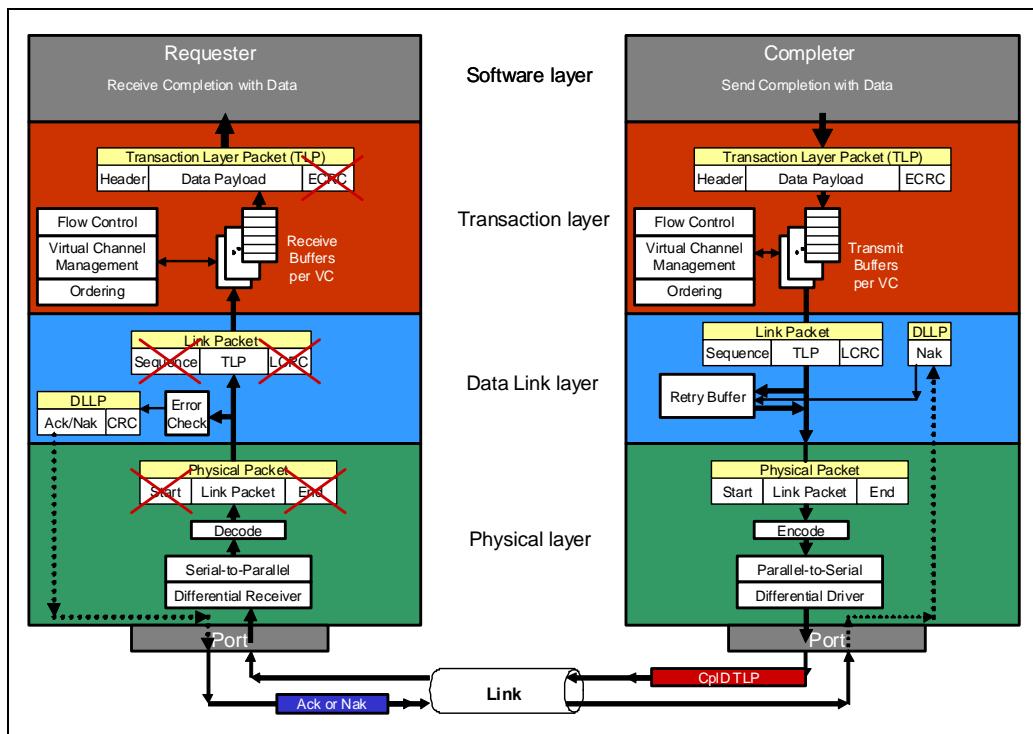
# Chapter 2: PCIe Architecture Overview

In addition to generating the Ack, the Completer's Link Layer also forwards the TLP up to its Transaction Layer. In the Completer's Transaction Layer, the TLP is placed in the appropriate VC receive buffer to be processed. An optional ECRC check can be performed, and if no error is found, the contents of the header (address, Requester ID, memory read transaction type, amount of data requested, traffic class etc.) are forwarded to the Completer's Software Layer.

## Completion with Data

For the second half of this discussion, refer to Figure 2-33 on page 83. To service the memory read request, the Completer Device Core/Software Layer sends a completion with data (CplD) request down to its Transaction Layer that includes the Requester ID and Tag copied from the original memory read request, transaction type, other parts of the completion header contents and the requested data.

Figure 2-33: Completion with Data Phase



# PCI Express Technology

---

The Transaction layer uses this information to build the CplD TLP, which always has a 3 DW header (it uses ID routing and never needs a 64-bit address). It also adds its own Completer ID to the header. This packet is also placed into the appropriate VC transmit buffer and, once selected, the flow control logic verifies that sufficient space is available at the neighboring device to receive this packet and, once confirmed, forwards the packet down to the Data Link Layer.

As before, the Data Link Layer adds a 12-bit Sequence Number and a 32-bit LCRC to the packet. A copy of the TLP with Sequence Number and LCRC is stored in the Replay Buffer and the packet is forwarded to the Physical Layer.

As before, the Physical Layer adds a Start and End character to the packet, byte stripes it across the available lanes, scrambles it, and 8b/10b encodes it. Finally, the CplD packet is serialized on all lanes and transmitted differentially across the Link to the neighbor.

The Requester converts the incoming serial bit stream back to 10-bit symbols and passes them through the elastic buffer. The 10-bit symbols are decoded back to bytes, de-scrambled and un-striped. The Start and End characters are detected and removed and the resultant TLP is sent up to the Data Link Layer.

As before, the Data Link Layer checks for LCRC errors in the received CplD TLP and checks the Sequence Number for missing or out-of-sequence TLPs. If there are no errors, it creates an Ack DLLP which contains the same Sequence Number as the CplD TLP used. A 16-bit CRC is added to the Ack DLLP and it's sent back to the Physical Layer which adds the proper framing symbols and transmits the Ack DLLP to the Completer.

The Completer Physical Layer checks and removes the framing symbols from the Ack DLLP and sends the remainder up to the Data Link Layer which checks the CRC. If there are no errors, it compares the Sequence Number with the Sequence Numbers for the TLPs stored in the Replay Buffer. The stored CplD TLP associated with the Ack received is recognized and that TLP is discarded from the Replay Buffer. If a Nak DLLP was received by the Completer instead, it would re-send a copy of the stored CplD TLP.

In the meantime, the Requester Transaction Layer receives the CplD TLP in the appropriate virtual channel buffer. Optionally, the Transaction layer can check for an ECRC error. If there are no errors, it forwards the header contents and data payload, including the Completion Status, to the Requester Software Layer, and we're done.

---

# 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-

# **PCI Express Technology**

---

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.

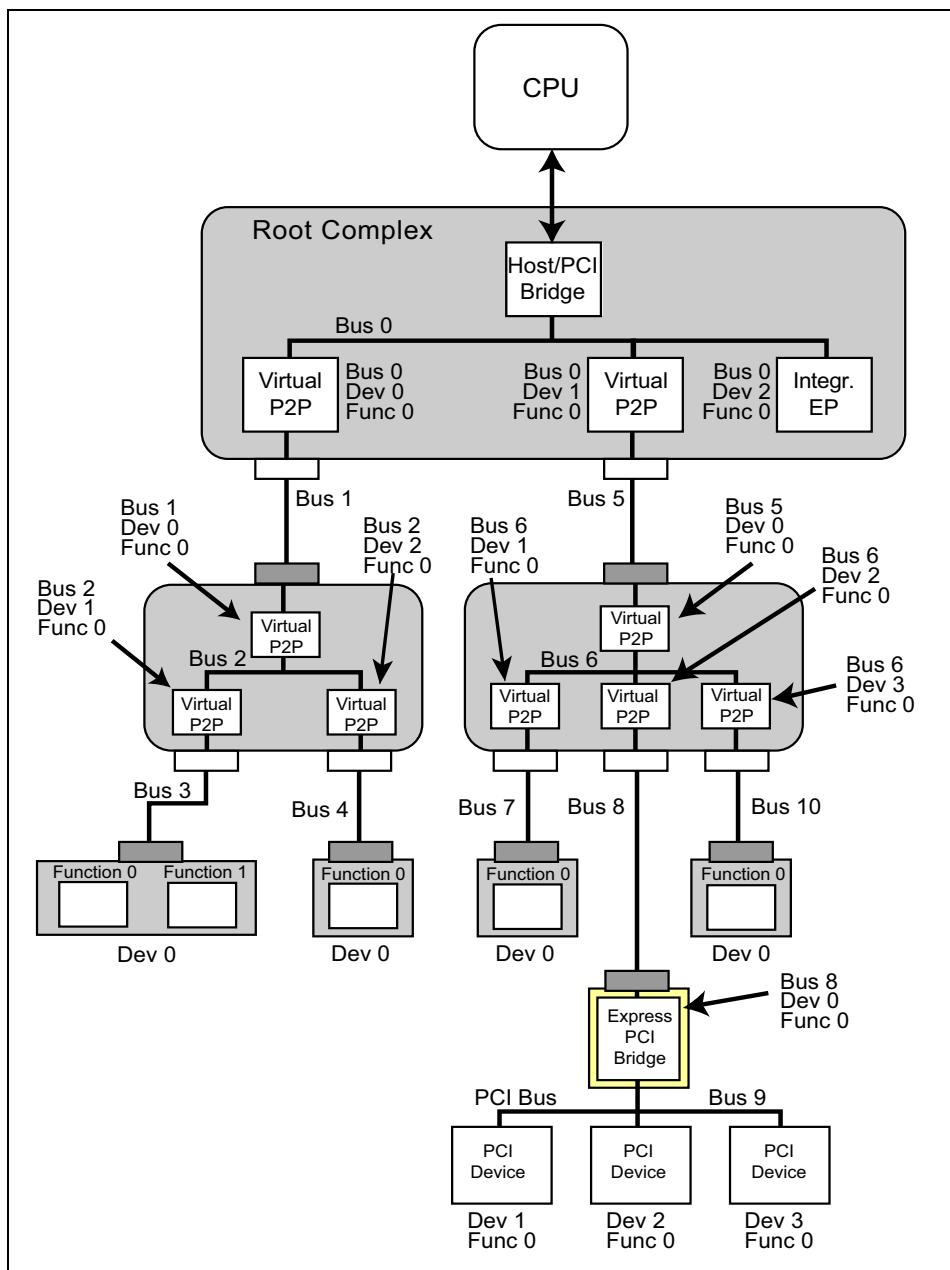
---

## **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.

## Chapter 3: Configuration Overview

*Figure 3-1: Example System*



# **PCI Express Technology**

---

## **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.

---

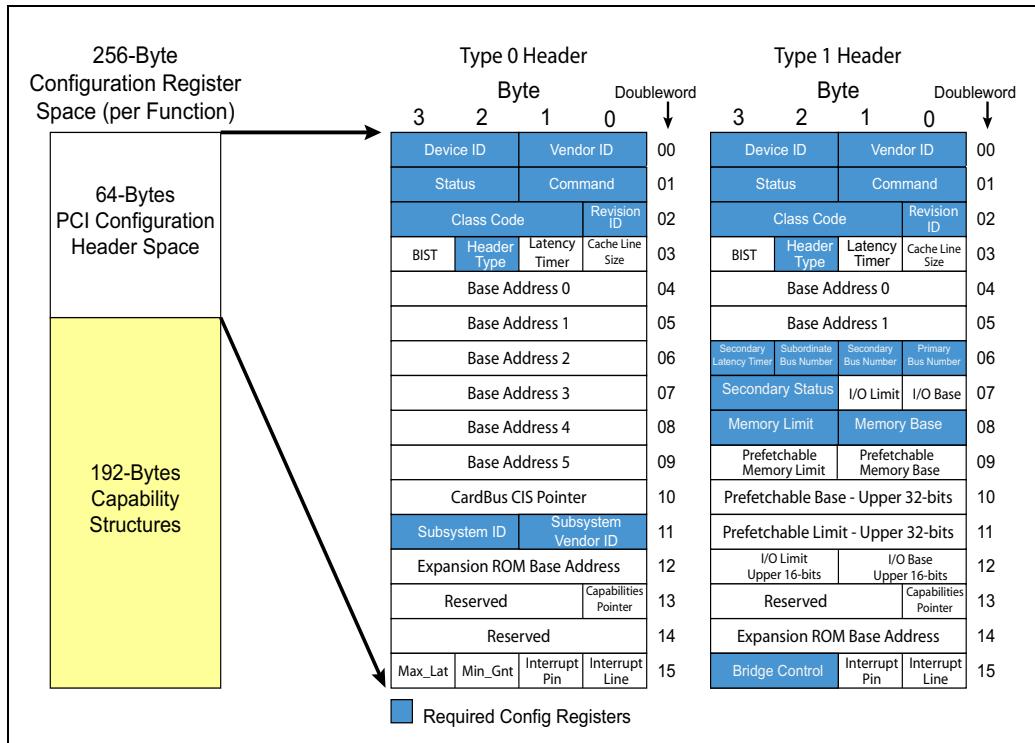
## **PCI-Compatible Space**

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 dwells (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 dwells 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

# Chapter 3: Configuration Overview

Figure 3-2: PCI Compatible Configuration Register Space

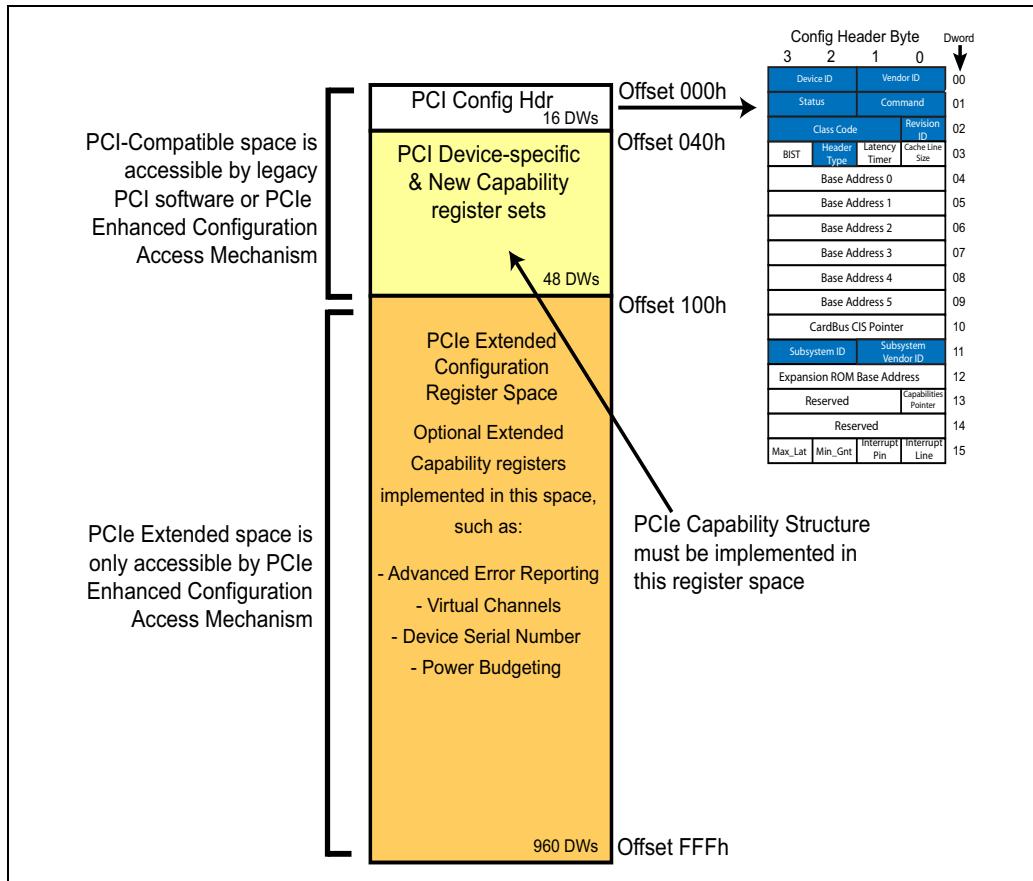


## Extended Configuration Space

Refer to Figure 3-3 on page 90 during this discussion. When PCIe was introduced, there was not enough room in the original 256-byte configuration region to contain all the new capability structures needed. So the size of configuration space was expanded from 256 bytes per function to 4KB, called the Extended Configuration Space. The 960-dword Extended Configuration area is only accessible using the Enhanced configuration mechanism and is therefore not visible to legacy PCI software. It contains additional optional Extended Capability registers for PCIe such as those listed in Figure 3-3 (not a complete list).

# PCI Express Technology

Figure 3-3: 4KB Configuration Space per PCI Express Function



## Host-to-PCI Bridge Configuration Registers

### General

The Host-to-PCI bridge's configuration registers don't have to be accessible using either of the configuration mechanisms mentioned in the previous section. Instead, it's typically implemented as device-specific registers in memory address space, which is known by the platform firmware. However, its configuration register layout and usage must adhere to the standard Type 0 template defined by the PCI 2.3 specification.

# **Chapter 3: Configuration Overview**

---

## **Only the Root Sends Configuration Requests**

The specification states that only the Root Complex is permitted to originate Configuration Requests. It acts as the system processor's liaison to inject Requests into the fabric and pass Completions back. The ability to originate configuration transactions is restricted to the processor through the Root Complex to avoid the anarchy that could result if any device had the ability to change the configuration of other devices.

Since only the Root can initiate these requests, they also can only move downstream, which means that peer-to-peer Configuration Requests are not allowed. The Requests are routed based on the target device's ID, meaning its BDF (Bus number in the topology, Device number on that bus, and Function number within that Device).

---

## **Generating Configuration Transactions**

Processors are generally unable to perform configuration read and write requests directly because they can only generate memory and IO requests. That means the Root Complex will need to translate certain of those accesses into configuration requests in support of this process. Configuration space can be accessed using either of two mechanisms:

- The legacy PCI configuration mechanism, using IO-indirect accesses.
- The enhanced configuration mechanism, using memory-mapped accesses.

---

## **Legacy PCI Mechanism**

The PCI spec defined an IO-indirect method for instructing the system (the Root Complex or its equivalent) to perform PCI configuration accesses. As it happened, the dominant PC processors (Intel x86) were only designed to address 64KB of IO address space. By the time PCI was defined, this limited IO space had become badly cluttered and only a few address ranges remained available: 0800h - 08FFh and 0C00h - 0CFFh. Consequently, it wasn't feasible to map the configuration registers for all the possible Functions directly into IO space. At the same time, memory address space was also limited in size and mapping all of configuration space into memory address space was not seen as a good solution either. So the spec writers chose a commonly-used solution to this problem, use indirect address mapping instead. To do this, one register holds the target

# PCI Express Technology

---

address, while a second holds the data going to or coming from the target. A write to the address register, followed by a read or write to the data register, causes a single read or write transaction to the correct internal address for the target function. This solves the problem of limited address space nicely, but it means that two IO accesses are needed to create one configuration access.

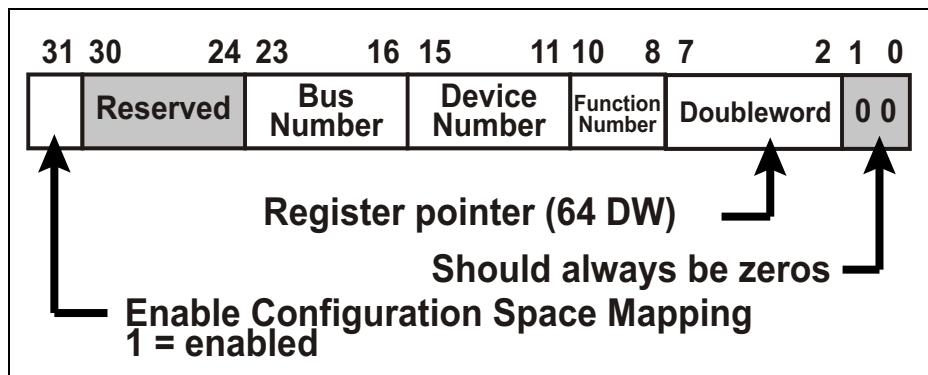
The PCI-Compatible mechanism uses two 32-bit IO ports in the Host bridge of the Root Complex. They are the **Configuration Address Port**, at IO addresses 0CF8h - 0CFBh, and the **Configuration Data Port**, at IO addresses 0CFCh - CFFh.

Accessing a Function's PCI-compatible configuration registers is accomplished by first writing the target Bus, Device, Function and dword numbers into the Configuration Address Port, setting its Enable bit in the process. Secondly, a one-, two-, or four-byte IO read or write is sent to the Configuration Data Port. The host bridge in the Root Complex compares the specified target bus to the range of buses that exist downstream of the bridge. If the target bus is within that range, the bridge initiates a configuration read or write request (depending on whether the IO access to the Configuration Data Port was a read or a write).

## Configuration Address Port

The Configuration Address Port only latches information when the processor performs a full 32-bit write to the port, as shown in Figure 3-4, and a 32-bit read from the port returns its contents. The information written to the Configuration Address Port must conform to the following template (illustrated in Figure 3-4) and described on the facing page.

Figure 3-4: Configuration Address Port at 0CF8h



# Chapter 3: Configuration Overview

---

- Bits [1:0] are hard-wired, read-only and must return **zeros** when read. The location is dword aligned and no byte-specific offset is allowed.
- Bits [7:2] identify the **target dword** (also called the Register Number) in the target Function's PCI-compatible configuration space. This mechanism is limited to the compatible configuration space (i.e., the first 64 doublewords of a Function's configuration space).
- Bits [10:8] identify the **target Function** number (0 - 7) within the target device.
- Bits [15:11] identify the **target Device** number (0 - 31).
- Bits [23:16] identify the **target Bus** number (0 - 255).
- Bits [30:24] are **reserved** and must be zero.
- Bit [31] must be set to 1b to enable translation of the subsequent IO access to the Configuration Data Port into a configuration access. If bit 31 is zero and an IO read or write is sent to the Configuration Data Port, the transaction is treated as an ordinary IO Request.

## Bus Compare and Data Port Usage

The Host Bridge within the Root Complex, shown in Figure 3-5 on page 95, implements a Secondary Bus Number register and a Subordinate Bus Number register. The Secondary Bus Number is the bus number of the bus immediately beneath the bridge. The Subordinate Bus Number is the target bus number that lives downstream of the bridge.

In a single Root Complex system, the bridge may have a Secondary Bus Number register that is hardwired to 0, a read/write register that reset forces to 0, or it may just implicitly know that the first accessible bus will be Bus 0. If bit 31 in the Configuration Address Port (see Figure 3-4 on page 92) is set to 1b, the bridge will compare the target bus number to the range of buses that exists downstream.

When a Request is seen, the Bridge evaluates whether the target bus number is within the range of bus numbers downstream, from the value of the Secondary Bus number to the Subordinate Bus number, inclusive. If the target bus matches the Secondary Bus, then that bus is targeted and the Request is passed through as a Type 0 Configuration Request. When devices see a Type 0 Request, they know that a device local to that bus is the target device (rather than one on a subordinate bus downstream).

If the target bus is larger than the bridge's Secondary Bus number, but less than or equal to the bridge's Subordinate Bus number, the Request will be forwarded as a Type 1 configuration request on the bridge's secondary bus. A Type 1 configuration access is understood to mean that, even though the Request has to go across this bus, it does not target a device on this bus. Instead, the request will

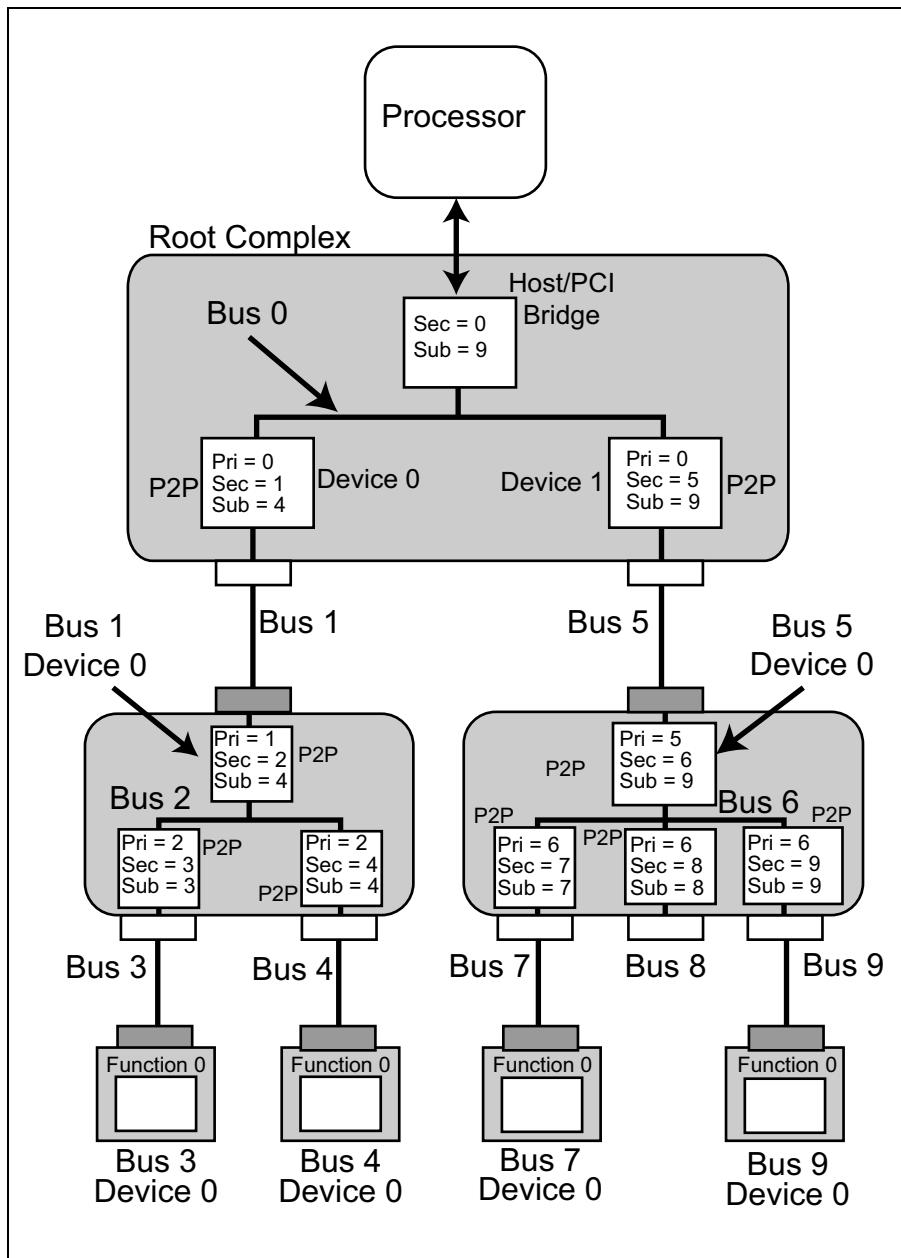
be forwarded downstream by one of the Bridges on this bus, whose Secondary and Subordinate bus number range contains the target bus number. For that reason, only Bridge devices pay attention to Type 1 configuration Requests. See “Configuration Requests” on page 99 for additional information regarding Type 0 and Type 1 configuration Requests.

## Single Host System

The information written to the Configuration Address Port is latched by the Host/PCI bridge within the Root Complex, as shown in Figure 3-1 on page 87. If bit 31 is 1b and the target bus is within the downstream range of bus numbers, the bridge translates a subsequent processor access targeting its Configuration Data Port into a configuration request on bus 0. The processor then initiates an IO read or write transaction to the Configuration Data Port at 0CFCh. This causes the bridge to generate a Configuration Request that is a read when the IO access to the Configuration Data Port was a read, or a Configuration write if the IO access was a write. It will be a Type 0 configuration transaction if the target bus is bus 0, or a Type 1 for another bus within the range, or not forwarded at all if the target bus is outside of the range.

## Chapter 3: Configuration Overview

Figure 3-5: Single-Root System



## Multi-Host System

If there are multiple Root Complexes (refer to Figure 3-6 on page 97), the Configuration Address and Data ports can be duplicated at the same IO addresses in each of their respective Host/PCI bridges. In order to prevent contention, only one of the bridges responds to the processor's accesses to the configuration ports.

1. When the processor initiates the IO write to the Configuration Address Port, the host bridges are configured so that only one will actively participate in the transaction.
2. During enumeration, software discovers and numbers all the buses under the active bridge. When that's done, it enables the inactive host bridge and assigns a bus number to it that is outside the range already assigned to the active bridge and continues the enumeration process. Both host bridges see the Requests, but since they have non-overlapping bus numbers they only respond to the appropriate bus number requests and so there's no conflict.
3. Accesses to the Configuration Address Port go to both host bridges after that, and a subsequent read or write access to the Configuration Data Port is only accepted by the host/PCI bridge that is the gateway to the target bus. This bridge responds to the processor's transaction and the other ignores it.
  - o If the target bus is the Secondary Bus, the bridge converts the access to a Type 0 configuration access.
  - o Otherwise, it converts it into a Type 1 configuration access.

---

## Enhanced Configuration Access Mechanism

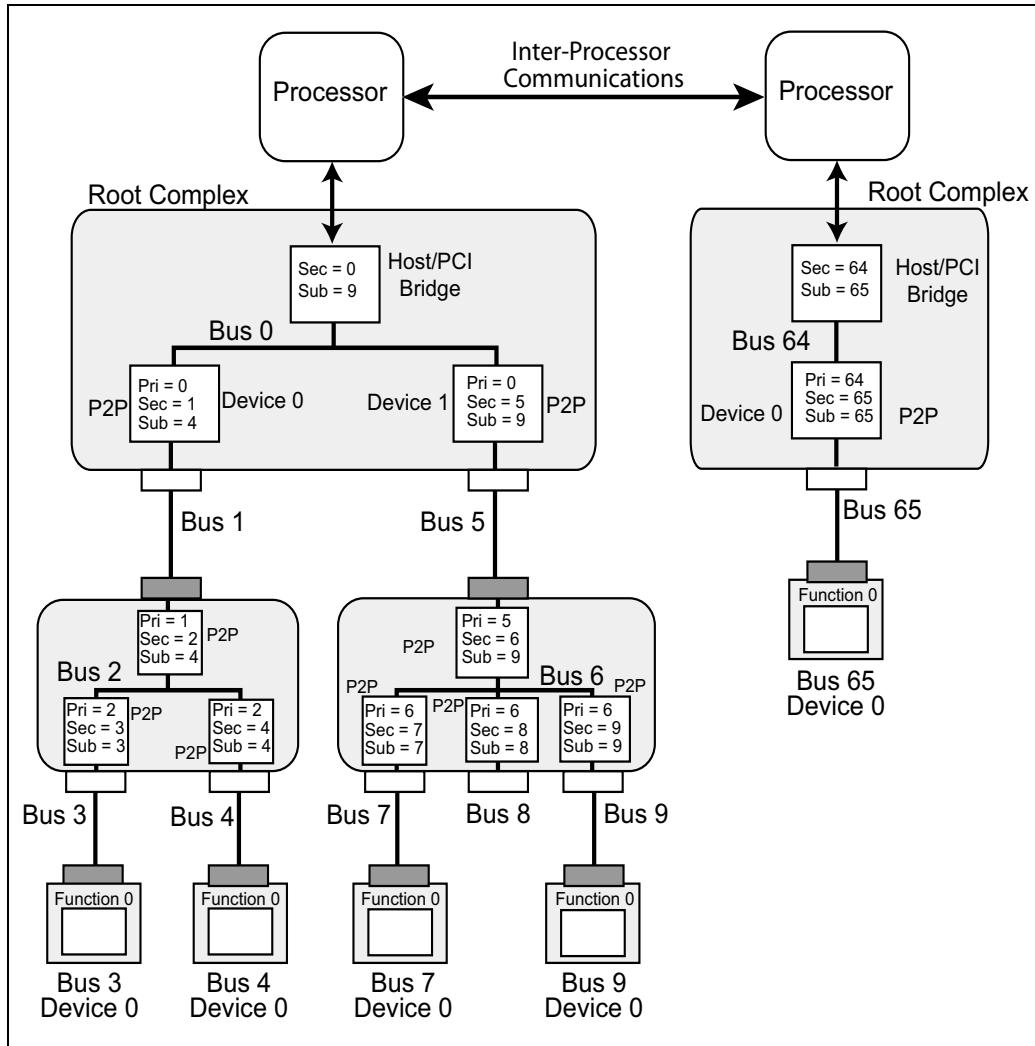
### General

When the spec writers were choosing how PCI-X and, later, PCIe, would access Configuration space, there were two concerns. First, the 256-byte space per Function limited vendors who wanted to put proprietary information there, as well as future spec writers who would need room for more standardized capability structures. To solve that problem, the space was simply extended from 256 bytes to 4KB per Function. Secondly, when PCI was developed there were few multi-processor systems in use. When there's only one CPU and it's only running one thread, the fact that the old model takes two steps to generate one access isn't a problem. But newer machines using multi-core, multi-threaded CPUs present a problem for the IO-indirect model because there's nothing to stop multiple threads from trying to access Configuration space at the same time. Consequently, the two-step model will no longer work without some locking semantics. With no locking semantics, once thread A writes a value into the

## Chapter 3: Configuration Overview

Configuration Address Port (CF8h), there is nothing to prevent thread B from overwriting that value before thread A can perform its corresponding access to the Configuration Data Port (CFCh).

Figure 3-6: Multi-Root System



# PCI Express Technology

---

To solve this new problem, the spec writers decided to take a different approach. Rather than try to conserve address space, they would create a single-step, uninterruptable process by mapping all of configuration space into memory addresses. That allows a single command sequence, since one memory request in the specified address range will generate one Configuration Request on the bus. The trade-off now is address size. Mapping 4KB per Function for all the possible implementations requires allocating 256MB of memory address space. The difference in that regard today is that modern architectures typically support anywhere between 36 and 48 bits of physical memory address space. With these memory address space sizes, 256MB is insignificant.

To handle this mapping, each Function's 4KB configuration space starts at a 4KB-aligned address within the 256MB memory address space set aside for configuration access, and the address bits now carry the identifying information about which Function is targeted (refer to Table 3-1 on page 98).

## Some Rules

A Root Complex is not required to support an access to enhanced configuration memory space if it crosses a dword address boundary (straddles two adjacent memory dwords). Nor are they required to support the bus locking protocol that some processor types use for an atomic, or uninterrupted series of commands. Software should avoid both of these situations when accessing configuration space unless it is known that the Root Complex does support them.

*Table 3-1: Enhanced Configuration Mechanism Memory-Mapped Address Range*

Memory Address Bit Field	Description
A[63:28]	Upper bits of the 256MB-aligned base address of the 256MB memory-mapped address range allocated for the Enhanced Configuration Mechanism. The manner in which the base address is allocated is implementation-specific. It is supplied to the OS by system firmware (typically through the ACPI tables).
A[27:20]	Target Bus Number (0 - 255).
A[19:15]	Target Device Number (0 - 31).
A[14:12]	Target Function Number (0 - 7).

## Chapter 3: Configuration Overview

---

Table 3-1: Enhanced Configuration Mechanism Memory-Mapped Address Range (Continued)

Memory Address Bit Field	Description
A[11:2]	A[11:2] this range can address one of 1024 dwords, whereas the legacy method is limited to only address one of 64 dwords.
A[1:0]	Defines the access size and the Byte Enable setting.

---

## Configuration Requests

Two request types, Type 0 or Type 1, may be generated by bridges in response to a configuration access. The type used depends on whether the target Bus number matches the bridge's Secondary Bus Number, as described below.

---

### Type 0 Configuration Request

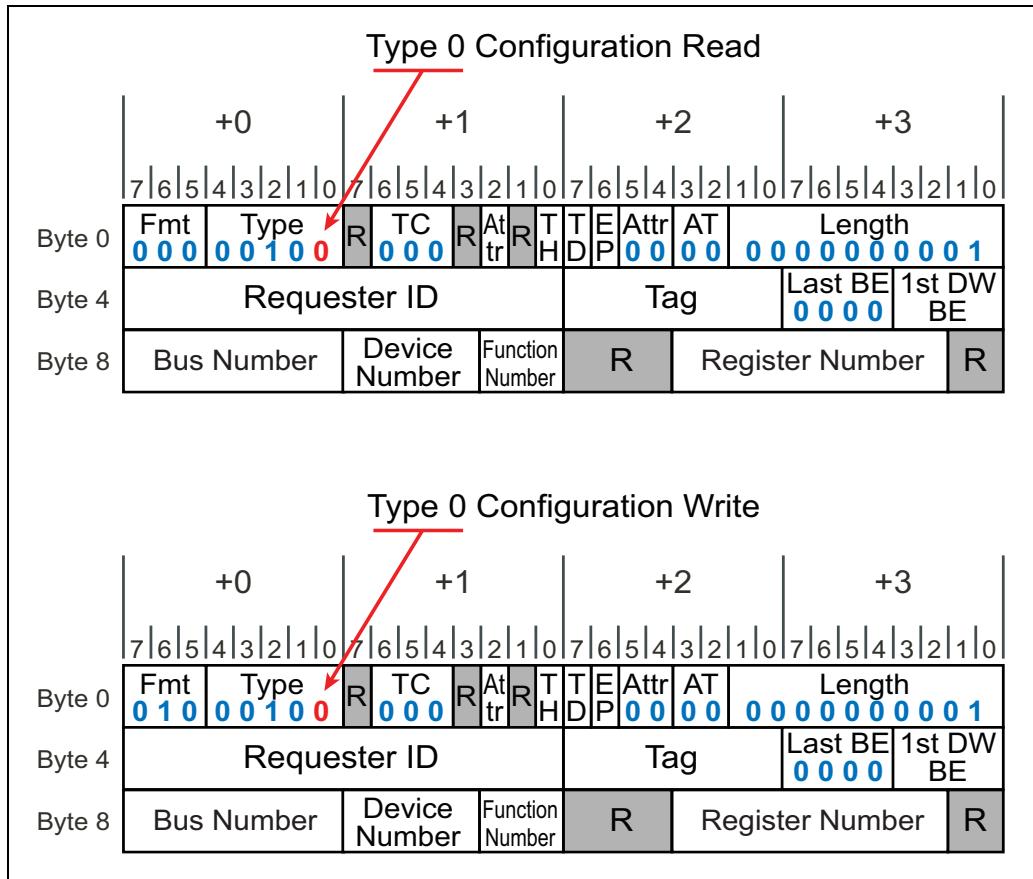
If the target bus number matches the Secondary Bus Number, a Type 0 configuration read or write is forwarded to the secondary bus and:

1. Devices on that Bus check the Device Number to see which of them is the target device. Note that Endpoints on an external Link will always be Device 0.
2. The selected Device checks the Function Number to see which Function is selected within the device.
3. The selected Function uses the Register Number field to select the target dword in its configuration space, and uses the First Dword Byte Enable field to select which bytes to read or write within the selected dword.

Figure 3-7 illustrates the Type 0 configuration read and write Request header formats. In both cases, the Type field = 00100, while the Format field indicates whether it's a read or a write.

# PCI Express Technology

Figure 3-7: Type 0 Configuration Read and Write Request Headers



## Type 1 Configuration Request

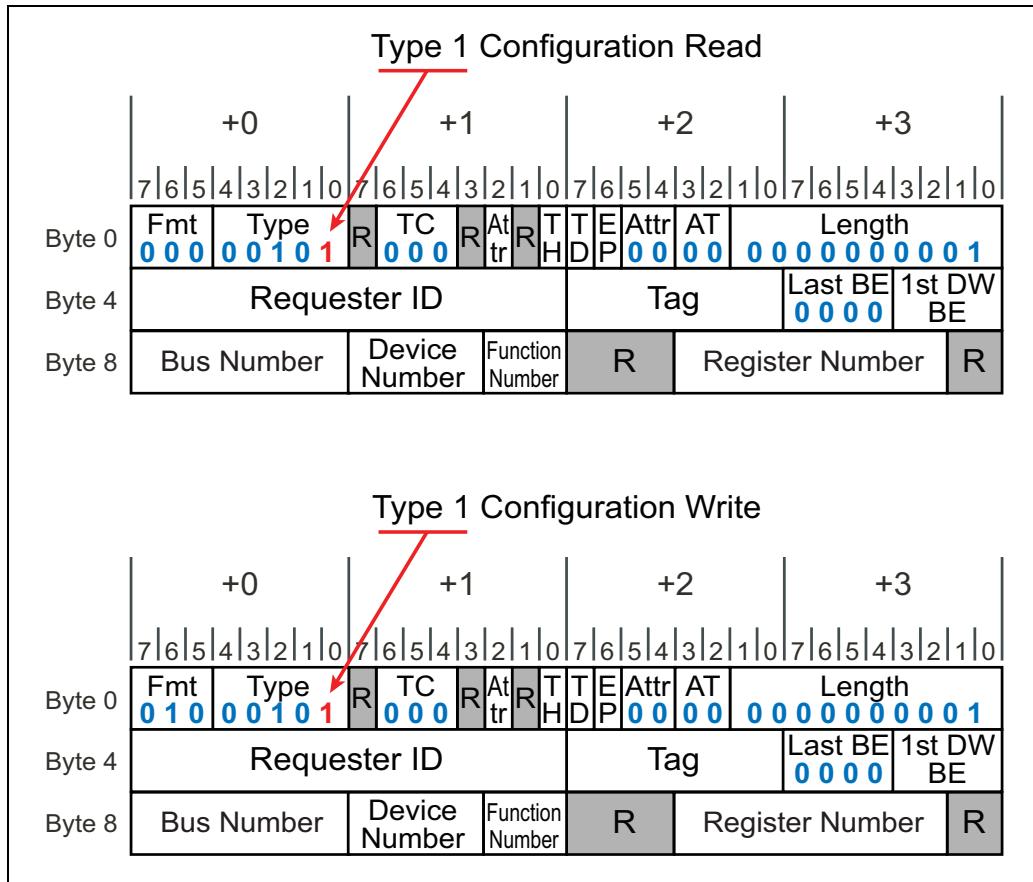
When a bridge sees a configuration access whose target bus number does not match its Secondary Bus Number but is in the range between its Secondary and Subordinate Bus Numbers, it forwards the packet as a Type 1 Request to its Secondary Bus. Devices that are not bridges (Endpoints) know to ignore Type 1 Requests since the target resides on a different bus, but bridges that see it will make the same comparison of the target bus number to the range of buses downstream (see Figure 3-1 on page 87 and Figure 3-6 on page 97).

# Chapter 3: Configuration Overview

- If the target bus matches the Bridge's secondary bus, the packet is converted from Type 1 to Type 0 and passed to the secondary bus. Devices local to that bus then check the packet header as previously described.
- If the target bus is not the Bridge's secondary bus but is within its range, the packet is forwarded to the Bridge's secondary bus as a Type 1 Request.

Figure 3-8 illustrates the Type 1 configuration read and write request header formats. In both cases, the Type field = 00101, while the Fmt field indicates whether it's a read or a write.

Figure 3-8: Type 1 Configuration Read and Write Request Headers



## Example PCI-Compatible Configuration Access

Refer to Figure 3-9 on page 104. To illustrate the concept of generating a Configuration Request using the legacy CF8h/CFCh mechanism, consider the following x86 assembly code sample, which will cause the Root Complex to perform a 2-byte read from Bus 4, Device 0, Function 0, Register 0 (Vendor ID).

```
mov    dx,0CF8h      ;set dx = config address port address
mov    eax,80040000h;enable=1, bus 4, dev 0, func 0, DW 0
out   dx,eax        ;IO write to set up address port
mov    dx,0CFCh      ; set dx = config data port address
in    ax,dx         ;2-byte read from config data port
```

1. The out instruction generates an IO write from the processor targeting the Configuration Address Port in the Root Complex Host bridge (0CF8h), as shown in Figure 3-4 on page 92.
2. The Host bridge compares the target bus number (4) specified in the Configuration Address Port to the range of buses (0-through-10) that reside downstream. The target bus falls within the range, so the bridge is primed with the destination of the next configuration request.
3. The in instruction generates an IO read transaction from the processor targeting the Configuration Data Port in the Root Complex Host bridge. It's a 2-byte read from the first two locations in the Configuration Data Port.
4. Since the target bus is not bus 0, the Host/PCI bridge initiates a Type 1 Configuration read on bus 0.
5. All of the devices on bus 0 latch the transaction request and see that it's a Type 1 Configuration Request. As a result, both of the virtual PCI-to-PCI bridges in the Root Complex compare the target bus number in the Type 1 request to the range of buses downstream from each of them.
6. The destination bus (4) is within the range of buses downstream of the left-hand bridge, so it passes the packet through to its secondary bus, but as a Type 1 request because the destination bus doesn't match the Secondary Bus Number.
7. The upstream port on the left-hand switch receives the packet and delivers it to the upstream PCI-to-PCI bridge.
8. The bridge determines that the destination bus resides beneath it, but is not targeting its secondary bus, so it passes the packet to bus 2 as a Type 1 request.
9. Both of the bridges on bus 2 receive the Type 1 request packet. The right-hand bridge determines that the destination bus matches its Secondary Bus Number.

## Chapter 3: Configuration Overview

---

10. The bridge passes the configuration read request through to bus 4, but converts into a Type 0 Configuration Read request because the packet has reached the destination bus (target bus number matches the secondary bus number).
11. Device 0 on bus 4 receives the packet and decodes the target Device, Function, and Register Number fields to select the target dword in its configuration space (see Figure 3-3 on page 90).
12. Bits 0 and 1 in the First Dword Byte Enable field are asserted, so the Function returns its first two bytes, (Vendor ID in this case) in the Completion packet. The Completion packet is routed to the Host bridge using the Requester ID field obtained from the Type 0 request packet.
13. The two bytes of read data are delivered to the processor, thus completing the execution of the “in” instruction. The Vendor ID is placed in the processor’s AX register.

---

### Example Enhanced Configuration Access

Refer to Figure 3-9 on page 104. The following x86 code sample causes the Root Complex to perform a read from Bus 4, Device 0, Function 0, Register 0 (Vendor ID). Before this will work, the Host Bridge must have been assigned a base address value. This example assumes that the 256MB-aligned base address of the Enhanced Configuration memory-mapped range is E0000000h:

```
mov ax, [E0400000h] ;memory-mapped Config read
```

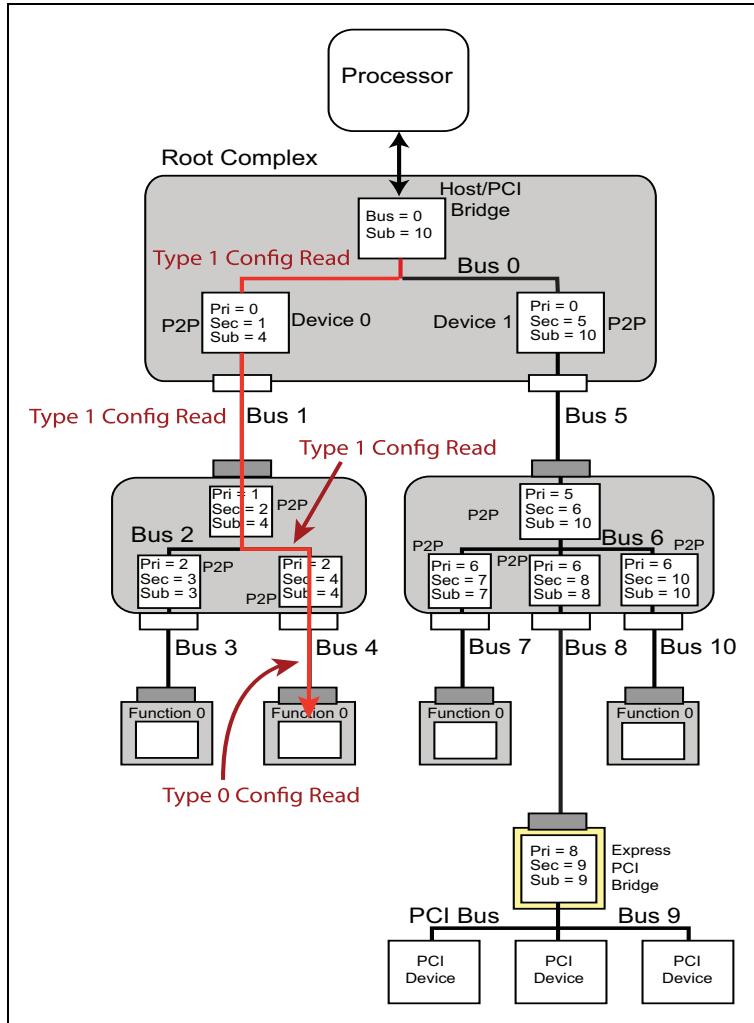
- Address bits 63:28 indicate the upper 36 bits of the 256MB-aligned base address of the overall Enhanced Configuration address range (in this case, 00000000 E0000000h).
- Address bits 27:20 select the target bus (in this case, 4).
- Address bits 19:15 select the target device (in this case, 0) on the bus.
- Address bits 14:12 select the target Function (in this case, 0) within the device.
- Address bits 11:2 selects the target dword (in this case, 0) within the selected Function’s configuration space.
- Address bits 1:0 define the start byte location within the selected dword (in this case, 0).

The processor initiates a 2-byte memory read starting from memory location E0400000h, and this is latched by the Host Bridge in the Root Complex. The Host Bridge recognizes that the address matches the area designated for Configuration and generates a Configuration read Request for the first two bytes in dword 0, Function 0, device 0, bus 4. The remainder of the operation is the same as that described in the previous section.

# PCI Express Technology

---

Figure 3-9: Example Configuration Read Access



---

## Enumeration - Discovering the Topology

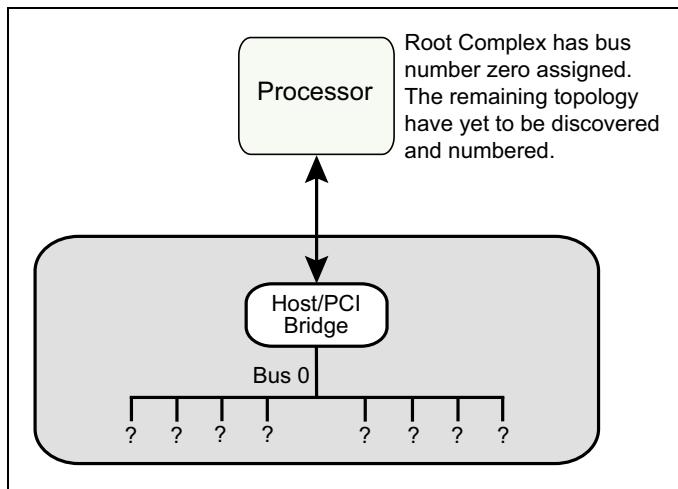
After a system reset or power up, configuration software has to scan the PCIe fabric to discover the machine topology and learn how the fabric is populated. Before that happens, as shown in Figure 3-10 on page 105, the only thing that software can know for sure is that there will be a Host/PCI bridge and that bus

## Chapter 3: Configuration Overview

---

number 0 will be on the secondary side of that bridge. Note that the upstream side of a bridge device is called its primary bus, while the downstream side is referred to as its secondary bus. The process of scanning the PCI Express fabric to discover its topology is referred to as the *enumeration process*.

Figure 3-10: Topology View At Startup



---

### Discovering the Presence or Absence of a Function

The configuration software executing on the processor normally discovers the existence of a Function by reading from its Vendor ID register. A unique 16-bit value is assigned to each vendor by the PCI-SIG and is hardwired into the Vendor ID register of each Function designed by that vendor. By reading this register in all of the possible combinations of Bus, Device, and Function numbers in the system, enumeration software can search through the entire topology to learn which devices are present. This process is fairly simple, but there are two problems that can arise: a targeted device may not be present, or it may be present but unprepared to respond. Handling these two cases is described next.

#### Device not Present

It can happen several times during the process of discovery that the targeted device doesn't actually exist in the system and when that happens it needs to be understood correctly. In PCI, the Configuration Read Request would timeout on the bus and generate a Master Abort error condition. Since no device was driving the bus and all the signals were pulled up, the data bits on the bus would be

seen as all ones and that would become the data value seen. The resulting Vendor ID of FFFFh is reserved. If enumeration software saw that result for the read, it understood that the device wasn't present. Since this wasn't really an error condition, the Master Abort would not be reported as an error during the enumeration process.

For PCIe, a Configuration Read Request to a non-existent device will result in the bridge above the target device returning a Completion without data that has a status of UR (Unsupported Request). For backward compatibility with the legacy enumeration model, the Root Complex returns all ones (FFFFh) to the processor for the data when this Completion is seen during enumeration. Note that enumeration software depends on receiving a value of all 1s for a Configuration Read Request that returns an Unsupported Request when probing for the existence of Functions in the system.

It's important to avoid accidentally reporting an error for this case. Even though this timeout or UR result would be seen as an error during runtime, it's an expected result that isn't considered an error during enumeration. To help avoid confusion on this, devices are usually not enabled to signal errors until later. For PCIe it may still be useful to make a note of this event, and that's why a fourth "error" status bit, called Unsupported Request Status is given in the PCIe Capability register block (refer to "Enabling/Disabling Error Reporting" on page 678 for more on this). That allows this condition to be noted without marking it as an error, and that's important because a detected error might stop the enumeration process to call the system error handler. The error handling software might have only limited capabilities during this time and thus have trouble resolving the problem. The enumeration software could fail in that case, since it's typically written to execute before the OS or other error handling software is available. To avoid this risk, errors should not normally be reported during enumeration.

## Device not Ready

Another problem that can arise is that the targeted device is present but isn't ready to respond to a configuration access. There is a timing consideration for configuration because of the time it takes devices to prepare for access. If the data rate is 5.0 GT/s or less, software must wait 100ms after reset before initiating a Configuration Request. If the rate is higher than 5.0 GT/s (Gen3 speed), software must wait until 100ms after Link training completes before attempting this. The reason for the longer delay for the higher speeds is that the Gen3 Equalization Process during Link training can take a long time (on the order of 50ms; see "Link Equalization Overview" on page 577 for more on this topic).

As defined in the PCI 2.3 spec, Initialization Time ( $T_{rhfa}$  - Time from Reset High to First Access) begins when RST# is deasserted and completes  $2^{25}$  PCI clocks

## Chapter 3: Configuration Overview

---

later. That works out to one full second during which the Function is preparing for its first configuration access and that value has been carried forward for PCIe as 1.0s (+50%/-0%). A Function could use that time to populate its configuration registers by loading the contents from an external serial EEPROM, for example. That might take a while to load and the Function would be unprepared for a successful access until it finished. In PCI, if a configuration access was seen before the Function was ready, it had three choices: ignore the Request, Retry the Request, or accept the Request but postpone delivering its response until it was fully ready. That last response could cause trouble for Hot-plug systems because the shared bus could end up being stalled for one second until the Request resolved.

In PCIe we have the same problem, but the process is a little different now. First, PCIe Functions must always give a Completion with a specific status when they are temporarily unable to respond to a configuration access, which is the Configuration Request Retry Status (CRS). This status is only legal in response to a configuration request and may optionally be considered a Malformed Packet error if seen in response to other Requests. This response is also only valid for the one second after reset because the Function is supposed to respond by then and can be considered broken if it won't.

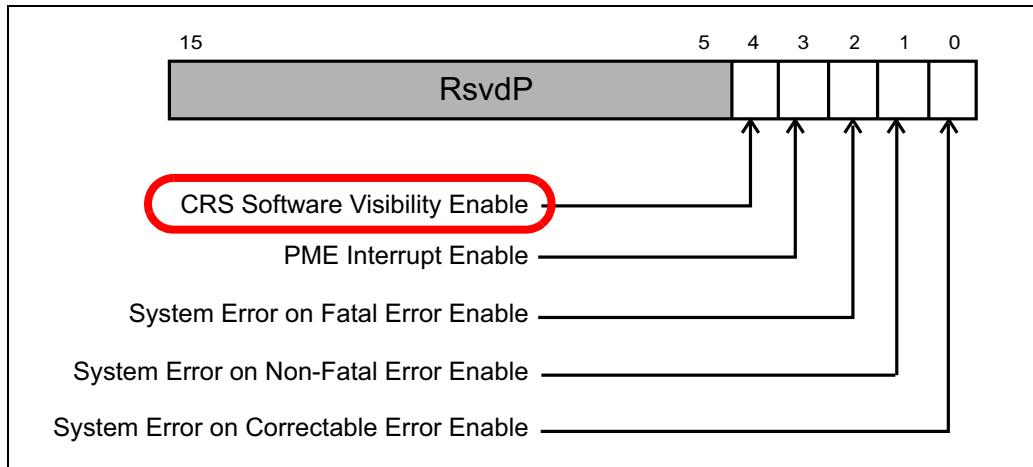
The way the Root Complex handles a CRS Completion in response to a Configuration Read Request is implementation specific, except for the period following a system reset. During that time, there are two options for what the Root will do next, based on the setting of the CRS Software Visibility bit in its Root Control Register, shown in Figure 3-11 on page 108:

- If the bit is set and the Request was a Configuration Read to both bytes of the Vendor ID register (as an enumeration access would do to discover the presence of a Function), the Root must give the host an artificial value of 0001h for this register, and all 1's for any additional bytes in this Request. This Vendor ID is not used for any real devices and will be interpreted by software as an indication of a potentially lengthy delay in accessing this device. This can be helpful because software could choose to go on to another task and make better use of the time that would otherwise be spent waiting for the device to respond, returning to query this device later. For this to work, software must ensure that its first access to a Function after a reset condition is a Configuration Read of both bytes of the Vendor ID.
- For configuration writes or any other configuration reads, the Root must automatically re-issue the Configuration Request again as a new request.

# PCI Express Technology

---

Figure 3-11: Root Control Register in PCIe Capability Block



---

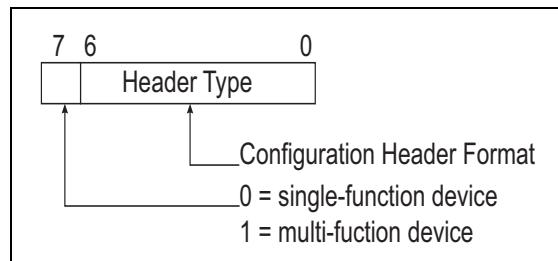
## Determining if a Function is an Endpoint or Bridge

A critical part of the enumeration process is being able to determine if a function is a bridge or an endpoint. As seen in Figure 3-12 on page 108, the lower 7 bits of the Header Type register (offset 0Eh in config space header) identify the basic category of the Function, and three values are defined:

- 0 = not a bridge (Endpoint in PCIe)
- 1 = PCI-to-PCI bridge (abbreviated as P2P) connecting two buses
- 2 = CardBus bridge (legacy interface not often used today)

In Figure 3-1 on page 87, the Header Type field (DW3, byte 2) in each of the Virtual P2Ps would return a value of 1, as would the PCI Express-to-PCI bridge (Bus 8, Device 0), while the Endpoints would return a Header Type of zero.

Figure 3-12: Header Type Register



## Single Root Enumeration Example

Now that we've discussed the basic elements involved in the enumeration process, let's walk through an example of the process. Figure 3-13 on page 113 illustrates an example system after the buses and devices have been enumerated. The discussion that follows assumes that the configuration software uses either of the two configuration access mechanisms defined in this chapter to achieve this result. At startup time, the configuration software executing on the processor performs enumeration as described below.

1. Software updates the Host/PCI bridge Secondary Bus Number to zero and the Subordinate Bus Number to 255. Setting this to the max value means that it won't have to be changed again until all the bus numbers downstream have been identified. For the moment, buses 0 through 255 are identified as being downstream.
2. Starting with Device 0 (bridge A), the enumeration software attempts to read the Vendor ID from Function 0 in each of the 32 possible devices on bus 0. If a valid Vendor ID is returned from Bus 0, Device 0, Function 0, the device exists and contains at least one Function. If not, go on to probe bus 0, device 1, Function 0.
3. The Header Type field in this example (Figure 3-12 on page 108) contains the value one (01h) indicating this is a PCI-to-PCI bridge. The Multifunction bit (bit 7) in the Header Type register is 0, indicating that Function 0 is the only Function in this bridge. *The spec doesn't preclude implementing multiple Functions within this Device and each of these Functions, in turn, could represent other virtual PCI-to-PCI bridges or even non-bridge functions.*
4. Now that software has found a bridge, performs a series of configuration writes to set the bridge's bus number registers as follows:
  - Primary Bus Number Register = 0
  - Secondary Bus Number Register = 1
  - Subordinate Bus Number Register = 255

The bridge is now aware that the number of the bus directly attached downstream is 1 (Secondary Bus Number = 1) and that the largest bus number downstream of it is 255 (Subordinate Bus Number = 255).

5. Enumeration software must perform a depth-first search. Before proceeding to discover additional Devices/Functions on bus 0, it must proceed to search bus 1.
6. Software reads the Vendor ID of Bus 1, Device 0, Function 0, which targets bridge C in our example. A valid Vendor ID is returned, indicating that Device 0, Function 0 exists on Bus 1.
7. The Header Type field in the Header register contains the value one (0000001b) indicating another PCI-to-PCI bridge. As before, bit 7 is a 0, indi-

# PCI Express Technology

---

- cating that bridge C is a single-function device.
8. Software now performs a series of configuration writes to set bridge C's bus number registers as follows:
    - Primary Bus Number Register = 1
    - Secondary Bus Number Register = 2
    - Subordinate Bus Number Register = 255
  9. Continuing the depth-first search, a read is performed from bus 2, device 0, Function 0's Vendor ID register. The example assumes that bridge D is Device 0, Function 0 on Bus 2.
  10. A valid Vendor ID is returned, indicating bus 2, device 0, Function 0 exists.
  11. The Header Type field in the Header register contains the value one (00000001b) indicating that this is a PCI-to-PCI bridge, and bit 7 is a 0, indicating that bridge D is a single-function device.
  12. Software now performs a series of configuration writes to set bridge D's bus number registers as follows:
    - Primary Bus Number Register = 2
    - Secondary Bus Number Register = 3
    - Subordinate Bus Number Register = 255
  13. Continuing the depth-first search, a read is performed from bus 3, device 0, Function 0's Vendor ID register.
  14. A valid Vendor ID is returned, indicating bus 3, device 0, Function 0 exists.
  15. The Header Type field in the Header register contains the value zero (00000000b) indicating that this is an Endpoint function. Since this is an endpoint and not a bridge, it has a Type 0 header and there are no PCI-compatible buses beneath it. This time, bit 7 is a 1, indicating that this is a multifunction device.
  16. Enumeration software performs accesses to the Vendor ID of all 8 possible functions in bus 3, device 0 and determines that only Function 1 exists in addition to Function 0. Function 1 is also an Endpoint (Type 0 header), so there are no additional buses beneath this device.
  17. Enumeration software continues scanning across on bus 3 to look for valid functions on devices 1 - 31 but does not find any additional functions.
  18. Having found every function there was to find downstream of bridge D, enumeration software updates bridge D, with the real Subordinate Bus Number of 3. Then it backs up one level (to bus 2) and continues scanning across on that bus looking for valid functions. The example assumes that bridge E is device 1, Function 0 on bus 2.
  19. A valid Vendor ID is returned, indicating that this Function exists.
  20. The Header Type field in bridge E's Header register contains the value one (00000001b) indicating that this is a PCI-to-PCI bridge, and bit 7 is a 0, indicating a single-function device.

## **Chapter 3: Configuration Overview**

---

21. Software now performs a series of configuration writes to set bridge E's bus number registers as follows:
  - Primary Bus Number Register = 2
  - Secondary Bus Number Register = 4
  - Subordinate Bus Number Register = 255
22. Continuing the depth-first search, a read is performed from bus 4, device 0, Function 0's Vendor ID register.
23. A valid Vendor ID is returned, indicating that this Function exists.
24. The Header Type field in the Header register contains the value zero (0000000b) indicating that this is an Endpoint device, and bit 7 is a 0, indicating that this is a single-function device.
25. Enumeration software scans bus 4 to look for valid functions on devices 1 - 31 but does not find any additional functions.
26. Having reached the bottom of this tree branch, enumeration software updates the bridge above that bus, E in this case, with the real Subordinate Bus Number of 4. It then backs up one level (to bus 2) and moves on to read the Vendor ID of the next device (device 2). The example assumes that devices 2 - 31 are not implemented on bus 2, so no additional devices are discovered on bus 2.
27. Enumeration software updates the bridge above bus 2, C in this case, with the real Subordinate Bus Number of 4 and backs up to the previous bus (bus 1) and attempts to read the Vendor ID of the next device (device 1). The example assumes that devices 1 - 31 are not implemented on bus 1, so no additional devices are discovered on bus 1.
28. Enumeration software updates the bridge above bus 1, A in this case, with the real subordinate Bus Number of 4. and backs up to the previous bus (bus 0) and moves on to read the Vendor ID of the next device (device 1). The example assumes that bridge B is device 1, function 0 on bus 0.
29. In the same manner as previously described, the enumeration software discovers bridge B and performs a series of configuration writes to set bridge B's bus number registers as follows:
  - Primary Bus Number Register = 0
  - Secondary Bus Number Register = 5
  - Subordinate Bus Number Register = 255
30. Bridge F is then discovered and a series of configuration writes are performed to set its bus number registers as follows:
  - Primary Bus Number Register = 5
  - Secondary Bus Number Register = 6
  - Subordinate Bus Number Register = 255
31. Bridge G is then discovered and a series of configuration writes are performed to set its bus number registers as follows:

# PCI Express Technology

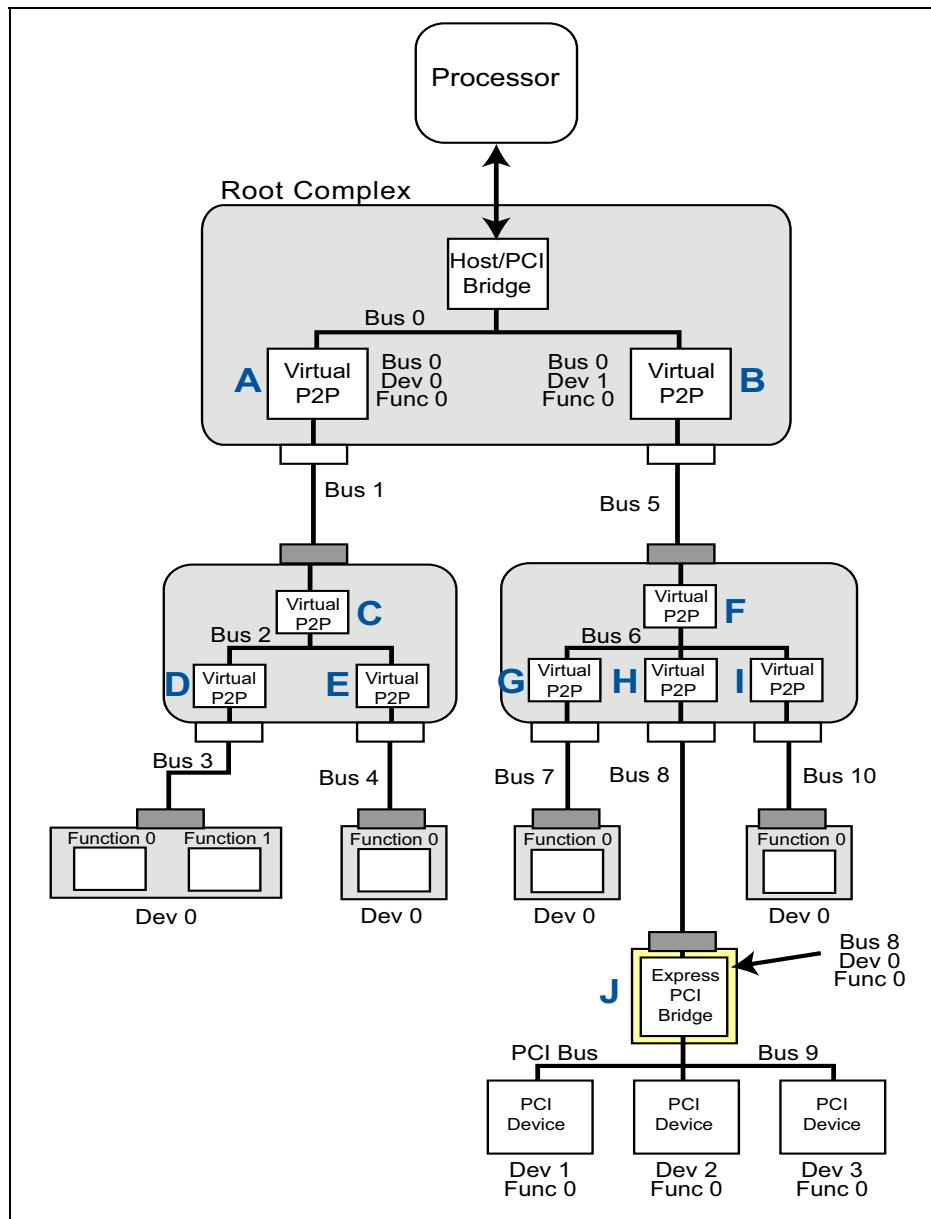
---

- Primary Bus Number Register = 6
  - Secondary Bus Number Register = 7
  - Subordinate Bus Number Register = 255
32. A single-function Endpoint device is discovered at bus 7, device 0, function 0, so the Subordinate Bus Number of Bridge G is updated to 7.
33. Bridge H is then discovered and a series of configuration writes are performed to set its bus number registers as follows:
- Primary Bus Number Register = 6
  - Secondary Bus Number Register = 8
  - Subordinate Bus Number Register = 255
34. Bridge J is discovered and a series of configuration writes are performed to set bridge its bus number registers as follows:
- Primary Bus Number Register = 8
  - Secondary Bus Number Register = 9
  - Subordinate Bus Number Register = 255
35. All devices and their respective Functions on bus 9 are discovered and none of them are bridges, so the Subordinate Bus Number of bridges H and J are updated to 9.
36. Bridge I is then discovered and a series of configuration writes are performed to set its bus number registers as follows:
- Primary Bus Number Register = 6
  - Secondary Bus Number Register = 10
  - Subordinate Bus Number Register = 255
37. A single-function Endpoint device is discovered at bus 10, device 0, function 0.
38. Since software has reached the bottom of this branch of the tree structure required for PCIe topologies, the Subordinate Bus Number registers for bridges B, F, and I are updated to 10, and so is the Host/PCI bridge's Subordinate Bus Number register.

The final values encoded into each bridge's Primary, Secondary and Subordinate Bus Number fields can be found in Figure 3-9 on page 104.

## Chapter 3: Configuration Overview

Figure 3-13: Single-Root System



---

## Multi-Root Enumeration Example

---

### General

Consider the Multi-Root System shown in Figure 3-14 on page 116. In this system, each Root Complex:

- Implements the Configuration Address Port and the Configuration Data Port at the same IO addresses (an x86-based system).
- Implements the Enhanced Configuration Mechanism.
- Contains a Host/PCI bridge.
- Implements the Secondary Bus Number and Subordinate Bus Number registers at separate addresses known to the configuration software.

In the illustration, each Root Complex is a chipset member and one of them is designated as the bridge to bus 0 (the primary Root Complex) while the other is designated as the bridge to bus 255 (secondary Root Complex).

---

### Multi-Root Enumeration Process

During enumeration of the left-hand tree structure in Figure 3-14 on page 116, the Host/PCI bridge in the secondary Root Complex ignores all configuration accesses because the targeted bus number is no greater than 9. Note that, although detected and numbered, Bus 8 has no device attached. Once that enumeration process has been completed, the enumeration software takes the following steps to enumerate the secondary Root Complex:

1. The enumeration software changes the Secondary and Subordinate Bus Number values in the secondary Root Complex's Host/PCI bridge to bus 64 in this example. (The values of 64 and 128 are commonly used as the starting bus number in multi-root systems, but this is just a software convention. There are no PCI or PCIe rules requiring that configuration. There would be nothing wrong with starting the secondary Root Complex's bus numbers at 10 in this example.)
2. Enumeration software then starts searching on bus 64 and discovers the bridge attached to the downstream Root Port.
3. A series of configuration writes are performed to set its bus number registers as follows:
  - Primary Bus Number Register = 64
  - Secondary Bus Number Register = 65
  - Subordinate Bus Number Register = 255

## **Chapter 3: Configuration Overview**

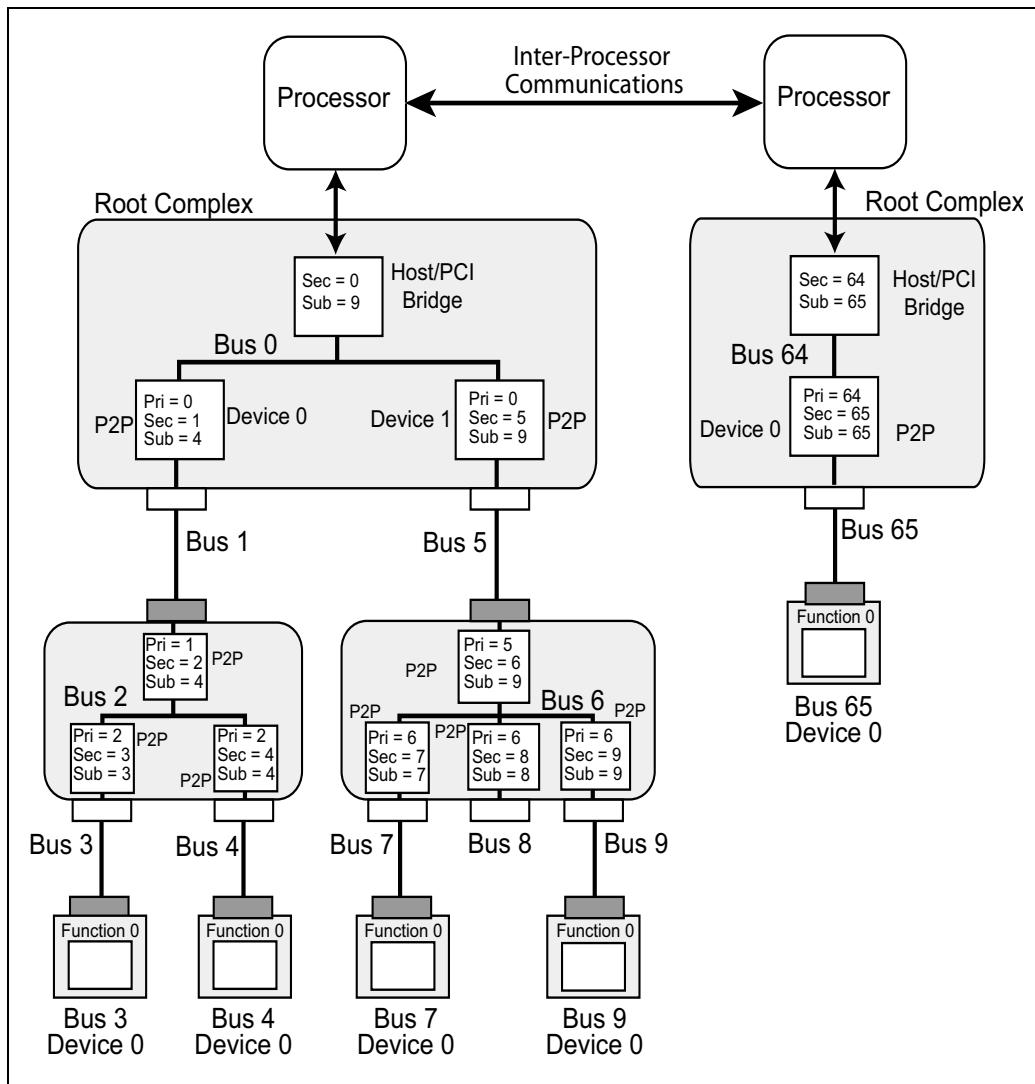
---

The bridge is now aware that the number of the bus directly attached to its downstream side is 65 (Secondary Bus Number = 65) and the number of the bus farthest downstream of it is 65 (Subordinate Bus Number = 65).

4. Device 0 is discovered on Bus 65 that implements only Function 0, and further searching reveals no other Devices are present on Bus 65, so the search process moves back up one Bus level.
5. Enumeration continues on bus 64 and no additional devices are discovered, so the Host/PCI's Subordinate Bus Number is updated to 65.
6. This completes the enumeration process.

# PCI Express Technology

Figure 3-14: Multi-Root System



## Hot-Plug Considerations

In a hot-plug environment, meaning one in which add-in cards can be added or removed during runtime, the situation illustrated by Bus number 8 in Figure 3-

## **Chapter 3: Configuration Overview**

---

14 on page 116 can potentially cause trouble. A problem can occur if the system has been enumerated and is up and running and then a card is plugged into Bus 8 that has a bridge on it. The bridge would need to have bus numbers assigned for its Secondary and Subordinate Bus Numbers that are higher than the bus number on its primary bus and completely inclusive. The reason is that the bus numbers have to be within the Secondary and Subordinate Bus Numbers of the bridge upstream of the new card.

One approach is to assign the Bus number(s) required for the bridge residing on Bus number 8 and increment the current Bus number 9 to a number that is one greater than the previous bus number, thereby making room for the new bus(s). Swizzling the bus numbers around during runtime can be done, but experienced people say it's hard to get it to work very well.

There is a simpler solution to this potential problem: simply leave a bus number gap whenever an unpopulated slot is found. For example, when Bus 8 is assigned but then an open slot is seen below it, give the next discovered bus a higher number, like 19 instead of 9, so as to leave room for these add-in situations to be resolved easily. Then, if a card with a bridge is added, the new bus number can be assigned as Bus 9 without causing any trouble. In most cases, leaving a bus number gap will not be an issue since the system can assign up to 256 bus numbers in total.

---

## **MindShare Arbor: Debug/Validation/Analysis and Learning Software Tool**

---

### **General**

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.

The book authors made a decision to not include detailed descriptions of all configuration registers summarized in a single chapter. Rather, registers are described throughout the book in associated chapters where they are relevant.

In lieu of a configuration register space description chapter in this book, MindShare Arbor is an excellent reference learning tool to quickly understand configuration registers and structures implemented in PCI, PCI-X and PCI Express

# PCI Express Technology

devices. All the register and field definitions are up-to-date with the latest version of the PCI Express spec. Several other types of structures (e.g. x86 MSRs, ACPI, USB, NVM Express) can also be viewed with MindShare Arbor (or will be coming soon).

Visit [www.mindshare.com/arbor](http://www.mindshare.com/arbor) to download a free trial version of MindShare Arbor.

Figure 3-15: Partial Screenshot of MindShare Arbor

The screenshot shows the MindShare Arbor software interface. The main window displays a tree view of system components under the heading "System (PCI View) - Scanned: Mon, 23 Jan 2012 13:22:59 CST". A table lists various PCI components with their device descriptions, device types, and capabilities. Below this, a specific component is selected: "0:1:0 - Intel Corporation 82G33/G31/P35/P31 Express PCI Express Root Port". This selection is shown in a detailed view with tabs for "register" and "raw".

**MSI Capability Structure (05h)**

MSL_CNTRL 00 00	NxtCapPtr A0	CapID 05	90h
Message Address [31:0] 00 00 00 00			94h
Reserved 00 00	Message Data 00 00		98h

**Link Capabilities**

31	24	23	22	21	20	19	18	17	15	14	12	11	10
0	0	0	0	0	1	0	0	0	0	0	1	0	0

**PCI Express Capability Structure (10h)**

Version 1

PCIe Capabilities 01 41	NxtCapPtr 00	CapID 10	A0h
Device Capabilities 00 00 80 00			A4h
Device Status 00 00	Device Control 00 00		A8h
Link Capabilities 02 01 25 01			ACh
Link Status 11 01	Link Control 00 40		B0h
Slot Capabilities 00 00 25 80			B4h
Slot Status 00 48	Slot Control 01 C0		B8h
Root Capabilities			

**3:0 - Max Link Speed (RO)**

Indicates the max link speed supported by this port.

- 0001b=2.5 GT/s
- 0010b=5.0 GT/s
- 0011b=8.0 GT/s

(Revision 3.0 of the base spec, made this field a binary indicated in the Link Capabilities 2 register.)

**9:4 - Maximum Link Width (RO)**

Indicates the max link width (number of lanes) supported.

- 000001b=x1
- 000010b=x2
- 000100b=x4
- 001000b=x8
- 001100b=x12
- 010000b=x16
- 100000b=x32

All other encodings are reserved.

**11:10 - Active State Power Management (ASPM) Support (RO)**

Indicates the level of ASPM supported on this link

- 00b=Reserved
- 01b=L0s only supported

# **Chapter 3: Configuration Overview**

---

## **MindShare Arbor Feature List**

- Description of all config registers included in the PCIe 3.0 spec
- Scan config space for all PCI-visible functions in system and a description of every one of these registers displayed in an easily readable format
- Directly access any memory or IO address
- Write to any config space location, memory address or IO address
- View standard and non-standard structures in a 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 your own XML-based decode files to drive Arbor's display
  - Create decode files for structures in config space, memory address space and IO space
- Save system scans for viewing later or on other systems
  - Saved system scans are XML-based and open-format
- New features that are either already in or coming soon:
  - Difference checking between scans
  - Post-processing scans for illegal or non-optimal settings
  - Scripting support for automation
  - Decode for x86 structures (MSRs, paging, segmentation, interrupt tables, etc.)
  - Decode for ACPI structures
  - Decode for USB structures
  - Decode for NVM Express structures

## **PCI Express Technology**

---

---

---

# **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

# PCI Express Technology

---

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

## Chapter 4: Address Space & Transaction Routing

---

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

## PCI Express Technology

---

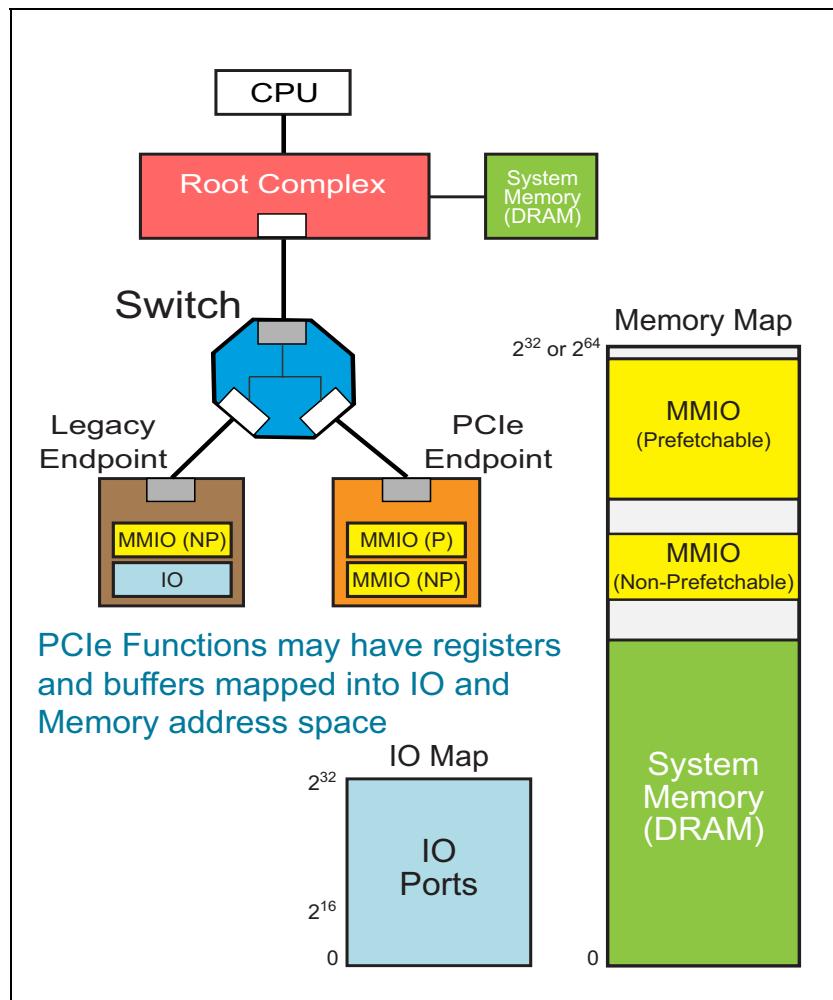
discard it to free up the buffer space. If the act of reading the data changed the value at that address (or had some other side effect), it would be impossible to recover the discarded data. However, for prefetchable space, the read had no side effects, so it is always possible to go back and get it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

## Chapter 4: Address Space & Transaction Routing

Figure 4-1: Generic Memory And IO Address Maps



---

## Base Address Registers (BARs)

---

### General

Each device in a system may have different requirements in terms of the amount and type of address space needed. For example, one device may have 256 bytes worth of internal registers/storage that should be accessible through IO address space and another device may have 16KB of internal registers/storage that should be accessible through MMIO.

PCI-based devices are not allowed to decide on their own, which addresses should be used to access their internal locations, that is the job of system software (i.e. BIOS and OS kernel). So the devices must provide a way for system software to determine the address space needs of the device. Once software knows what the device's requirements are in terms of address space, then assuming the request can be fulfilled, software will simply allocate an available range of addresses, of the appropriate type (IO, NP-MMIO or P-MMIO), to that device.

This is all accomplished through the Base Address Registers (BARs) in the header of configuration space. As shown in Figure 4-2 on page 127, a Type 0 header has six BARs available (each one being 32 bits in size), while a Type 1 header has only two BARs. Type 1 headers are found in all bridge devices, which means every switch port and root complex port has a Type 1 header. Type 0 headers are in non-bridge devices like endpoints. An example of this can be seen in Figure 4-3 on page 128.

System software must first determine the size and type of address space being requested by a device. The device designer knows the collective size of the internal registers/storage that should be accessible via IO or MMIO. The device designer also knows how the device will behave when those registers are accessed (i.e. do reads have side-effects or not). This will determine whether prefetchable MMIO (reads have no side-effects) or non-prefetchable MMIO (reads do have side-effects) should be requested. Knowing this information, the device designer hard-codes the lower bits of the BARs to certain values indicating the type and size of the address space being requested.

The upper bits of the BARs are writable by software. Once system software checks the lower bits of the BARs to determine the size and type of address space requested, system software will then write the base address of the address range being allocated to this device into the upper bits of the BAR. Since a single

# Chapter 4: Address Space & Transaction Routing

Endpoint (Type 0 header) has six BARs, up to six different address space requests can be made. However, this is not common in the real world. Most devices will request 1-3 different address ranges.

Not all BARs have to be implemented. If a device does not need all the BARs to map their internal registers, the extra BARs are hard-coded with all 0's notifying software that these BARs are not implemented.

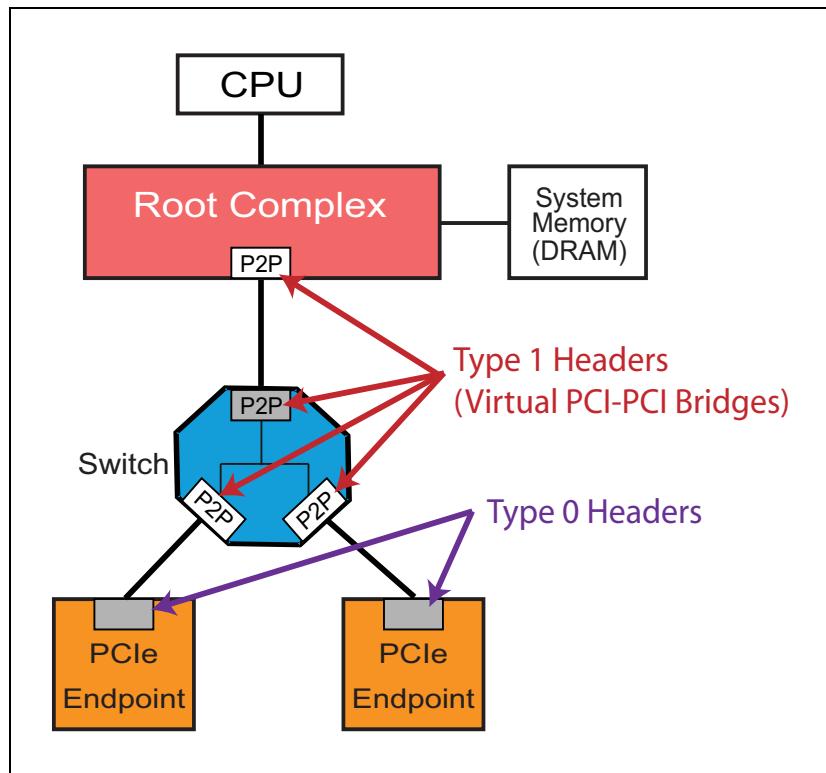
Figure 4-2: BARs in Configuration Space

Type 0 Header				Type 1 Header								
31	23	15	7 0	00h	00h	00h	00h					
Device ID		Vendor ID		04h	Device ID		04h					
Status		Command		08h	Status		08h					
Class Code			Rev ID	0Ch	Class Code							
BIST	Header Type	Latency Timer	Cache Line Size	10h	Base Address 0 (BAR0)							
Base Address 0 (BAR0)				14h	Base Address 1 (BAR1)							
Base Address 1 (BAR1)				18h	Base Address 2 (BAR2)							
Base Address 2 (BAR2)				1Ch	Base Address 3 (BAR3)							
Base Address 3 (BAR3)				20h	Base Address 4 (BAR4)							
Base Address 4 (BAR4)				24h	Base Address 5 (BAR5)							
Base Address 5 (BAR5)				28h	CardBus CIS Pointer							
Subsystem Device ID		Subsystem Vendor ID		2Ch	Subsystem Device ID							
Expansion ROM Base Address				30h	Expansion ROM Base Address							
Reserved			Capability Pointer	34h	IO Limit Upper 16 Bits			30h				
Reserved				38h	IO Base Upper 16 Bits			34h				
Max Lat	Min Gnt	Interrupt Pin	Interrupt Line	3Ch	IO Limit Upper 32 Bits			38h				
Bridge Control				IO Base Upper 32 Bits				3Ch				
Interrupt Pin				Non-Prefetchable Memory Limit				Bridge Control				
Interrupt Line				Prefetchable Memory Limit				Interrupt Pin				
Expansion ROM Base Address				Non-Prefetchable Memory Base				Interrupt Line				
Bridge Control				Prefetchable Memory Base				Expansion ROM Base Address				
Interrupt Pin				Prefetchable Memory Limit				Bridge Control				
Interrupt Line				IO Limit Upper 16 Bits				Interrupt Pin				
Expansion ROM Base Address				IO Base Upper 16 Bits				Expansion ROM Base Address				
Bridge Control				IO Limit Upper 32 Bits				Bridge Control				
Interrupt Pin				IO Base Upper 32 Bits				Interrupt Pin				
Interrupt Line				Non-Prefetchable Memory Base				Expansion ROM Base Address				
Expansion ROM Base Address				Prefetchable Memory Base				Bridge Control				
Bridge Control				Prefetchable Memory Limit				Interrupt Pin				
Interrupt Pin				IO Limit Upper 16 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				IO Base Upper 16 Bits				Bridge Control				
Bridge Control				IO Limit Upper 32 Bits				Interrupt Pin				
Interrupt Pin				IO Base Upper 32 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				Non-Prefetchable Memory Base				Bridge Control				
Bridge Control				Prefetchable Memory Base				Interrupt Pin				
Interrupt Pin				Prefetchable Memory Limit				Expansion ROM Base Address				
Expansion ROM Base Address				IO Limit Upper 16 Bits				Bridge Control				
Bridge Control				IO Base Upper 16 Bits				Interrupt Pin				
Interrupt Pin				IO Limit Upper 32 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				IO Base Upper 32 Bits				Bridge Control				
Bridge Control				Non-Prefetchable Memory Base				Interrupt Pin				
Interrupt Pin				Prefetchable Memory Base				Expansion ROM Base Address				
Expansion ROM Base Address				Prefetchable Memory Limit				Bridge Control				
Bridge Control				IO Limit Upper 16 Bits				Interrupt Pin				
Interrupt Pin				IO Base Upper 16 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				IO Limit Upper 32 Bits				Bridge Control				
Bridge Control				IO Base Upper 32 Bits				Interrupt Pin				
Interrupt Pin				Non-Prefetchable Memory Base				Expansion ROM Base Address				
Expansion ROM Base Address				Prefetchable Memory Base				Bridge Control				
Bridge Control				Prefetchable Memory Limit				Interrupt Pin				
Interrupt Pin				IO Limit Upper 16 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				IO Base Upper 16 Bits				Bridge Control				
Bridge Control				IO Limit Upper 32 Bits				Interrupt Pin				
Interrupt Pin				IO Base Upper 32 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				Non-Prefetchable Memory Base				Bridge Control				
Bridge Control				Prefetchable Memory Base				Interrupt Pin				
Interrupt Pin				Prefetchable Memory Limit				Expansion ROM Base Address				
Expansion ROM Base Address				IO Limit Upper 16 Bits				Bridge Control				
Bridge Control				IO Base Upper 16 Bits				Interrupt Pin				
Interrupt Pin				IO Limit Upper 32 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				IO Base Upper 32 Bits				Bridge Control				
Bridge Control				Non-Prefetchable Memory Base				Interrupt Pin				
Interrupt Pin				Prefetchable Memory Base				Expansion ROM Base Address				
Expansion ROM Base Address				Prefetchable Memory Limit				Bridge Control				
Bridge Control				IO Limit Upper 16 Bits				Interrupt Pin				
Interrupt Pin				IO Base Upper 16 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				IO Limit Upper 32 Bits				Bridge Control				
Bridge Control				IO Base Upper 32 Bits				Interrupt Pin				
Interrupt Pin				Non-Prefetchable Memory Base				Expansion ROM Base Address				
Expansion ROM Base Address				Prefetchable Memory Base				Bridge Control				
Bridge Control				Prefetchable Memory Limit				Interrupt Pin				
Interrupt Pin				IO Limit Upper 16 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				IO Base Upper 16 Bits				Bridge Control				
Bridge Control				IO Limit Upper 32 Bits				Interrupt Pin				
Interrupt Pin				IO Base Upper 32 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				Non-Prefetchable Memory Base				Bridge Control				
Bridge Control				Prefetchable Memory Base				Interrupt Pin				
Interrupt Pin				Prefetchable Memory Limit				Expansion ROM Base Address				
Expansion ROM Base Address				IO Limit Upper 16 Bits				Bridge Control				
Bridge Control				IO Base Upper 16 Bits				Interrupt Pin				
Interrupt Pin				IO Limit Upper 32 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				IO Base Upper 32 Bits				Bridge Control				
Bridge Control				Non-Prefetchable Memory Base				Interrupt Pin				
Interrupt Pin				Prefetchable Memory Base				Expansion ROM Base Address				
Expansion ROM Base Address				Prefetchable Memory Limit				Bridge Control				
Bridge Control				IO Limit Upper 16 Bits				Interrupt Pin				
Interrupt Pin				IO Base Upper 16 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				IO Limit Upper 32 Bits				Bridge Control				
Bridge Control				IO Base Upper 32 Bits				Interrupt Pin				
Interrupt Pin				Non-Prefetchable Memory Base				Expansion ROM Base Address				
Expansion ROM Base Address				Prefetchable Memory Base				Bridge Control				
Bridge Control				Prefetchable Memory Limit				Interrupt Pin				
Interrupt Pin				IO Limit Upper 16 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				IO Base Upper 16 Bits				Bridge Control				
Bridge Control				IO Limit Upper 32 Bits				Interrupt Pin				
Interrupt Pin				IO Base Upper 32 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				Non-Prefetchable Memory Base				Bridge Control				
Bridge Control				Prefetchable Memory Base				Interrupt Pin				
Interrupt Pin				Prefetchable Memory Limit				Expansion ROM Base Address				
Expansion ROM Base Address				IO Limit Upper 16 Bits				Bridge Control				
Bridge Control				IO Base Upper 16 Bits				Interrupt Pin				
Interrupt Pin				IO Limit Upper 32 Bits				Expansion ROM Base Address				
Expansion ROM Base Address				IO Base Upper 32 Bits				Bridge Control				
Bridge Control				Non-Prefetchable Memory Base				Interrupt Pin				
Interrupt Pin												

# PCI Express Technology

---

Figure 4-3: PCI Express Devices And Type 0 And Type 1 Header Use



---

## BAR Example 1: 32-bit Memory Address Space Request

Figure 4-4 on page 130 shows the basic steps in setting up a BAR, which in this example, is requesting a 4KB block of non-prefetchable memory (NP-MMIO). In the figure, the BAR is shown at three points in the configuration process:

1. In (1) of Figure 4-4, we see the uninitialized state of the BAR. The device designer has fixed the lower bits to indicate the size and type, but the upper bits (which are read-write) are shown as Xs to indicate their value is not known. System software will first write all 1s to every BAR (using config writes) to set all writable bits. (Of course, the hard-coded lower bits are unaffected by any configuration writes.) The second view of the BAR,

## Chapter 4: Address Space & Transaction Routing

---

shown in (2) of Figure 4-4, shows how it looks after configuration software has written all 1's to it.

Writing all 1s is done to determine what the least-significant writable bit is. This bit position indicates the size of the address space being requested. In this example, the least-significant writable bit is bit 12, so this BAR is requesting  $2^{12}$  (or 4KB) of address space. If the least significant writable bit would have been bit 20, then the BAR would have been requesting  $2^{20}$  (or 1MB) of address space.

2. After writing all 1s to the BARs, software turns around and reads the value of each BAR, starting with BAR0, to determine the type and size of the address space being requested. Table 4-1 on page 129 summarizes the results of the configuration read of BAR0 for this example.
3. The final step in this process is for system software to allocate an address range to BAR0 now that software knows the size and type of the address space being requested. The third view of the BAR, in (3) of Figure 4-4, shows how it looks after software has written the start address for the allocated block of addresses. In this example, the start address is F900\_0000h.

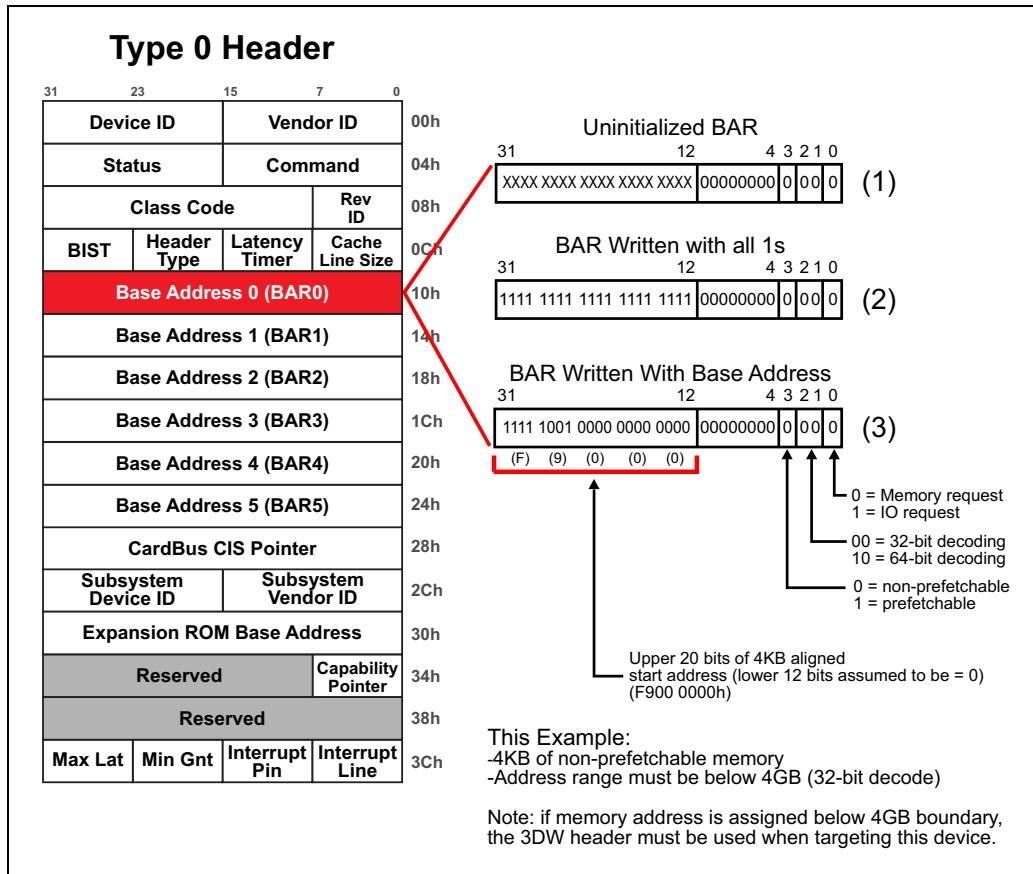
At this point the configuration of BAR0 is complete. Once software enables memory address decoding in the Command register (offset 04h), this device will accept any memory requests it receives that fall within the range from F900\_0000h - F900\_0FFFh (4KB in size).

*Table 4-1: Results of Reading the BAR after Writing All 1s To It*

BAR Bits	Meaning
0	Read as 0b, indicating a memory request. Since this is a memory request, bits 3:1 also have an encoded meaning.
2:1	Read as 00b indicating the target only supports decoding a 32-bit address
3	Read as 0b, indicating request is for non-prefetchable memory (meaning reads do have side-effects); NP-MMIO
11:4	Read as all 0s, indicating the size of the request (these bits are hard-coded to 0)
31:12	Read as all 1s because software has not yet programmed the upper bits with a start address for the block. Since bit 12 is the least significant bit that could be written, the memory size requested is $2^{12} = 4\text{KB}$ .

# PCI Express Technology

Figure 4-4: 32-Bit Non-Prefetchable Memory BAR Set Up



## BAR Example 2: 64-bit Memory Address Space Request

In the previous example, we saw BAR0 being used to request non-prefetchable memory address space (NP-MMIO). In this example, as shown in Figure 4-5 on page 132, BAR1 and BAR2 are being used to request a 64MB block of prefetchable memory address space. Two sequential BARs are being used here because the device supports a 64-bit address for this request, meaning that software can allocate the requested address space above the 4GB address boundary if it

## Chapter 4: Address Space & Transaction Routing

---

wants to (but that is not a requirement). Since the address can be a 64-bit address, two sequential BARs must be used together.

As before, the BARs are shown at three points in the configuration process:

1. In (1) of Figure 4-5, we see the uninitialized state of the BAR pair. The device designer has hard-coded the lower bits of the lower BAR (BAR1 in our example) to indicate the request type and size, while the bits of the upper BAR (BAR2) are all read-write. System software's first step was to write all 1s to every BAR. In (2) of Figure 4-5, we see the BARs after having all 1s written to them.
2. As described in the previous example, system software already evaluated BAR0. So software's next step is to read the next BAR (BAR1) and evaluate it to see if the device is requesting additional address space. Once BAR1 is read, software realizes that more address space is being requested and this request is for prefetchable memory address space that can be allocated anywhere in the 64-bit address range. Since it supports a 64-bit address, the next sequential BAR (BAR2 in this case) is treated as the upper 32 bits of BAR1. So software now also reads in the contents of BAR2. However, software does not evaluate the lower bits of BAR2 in the same way it did for BAR1, because it knows BAR2 is simply the upper 32 bits of the 64-bit address request started in BAR1. Table 4-2 on page 132 summarizes the results of these configuration reads.
3. The final step in this process is for system software to allocate an address range to the BARs now that software knows the size and type of the address space being requested. The third view of the BARs in (3) of Figure 4-5 shows the result after software has used two configuration writes to program the 64-bit start address for the allocated range. In this example, bit 1 of the Upper BAR (address bit 33 in the BAR pair) is set and bit 30 of the Lower BAR (address bit 30 in the BAR pair) is set to indicate a start address of  $2_{-}4000_{-}0000h$ . All other writable bits in both BARs are cleared.

At this point, the configuration of the BAR pair (BAR1 & BAR2) is complete. Once software enables memory address decoding in the Command register (offset 04h), this device will accept any memory requests it receives that fall within the range from  $2_{-}4000_{-}0000h$  -  $2_{-}43FF_{-}FFFFh$  (64MB in size).

# PCI Express Technology

---

Figure 4-5: 64-Bit Prefetchable Memory BAR Set Up

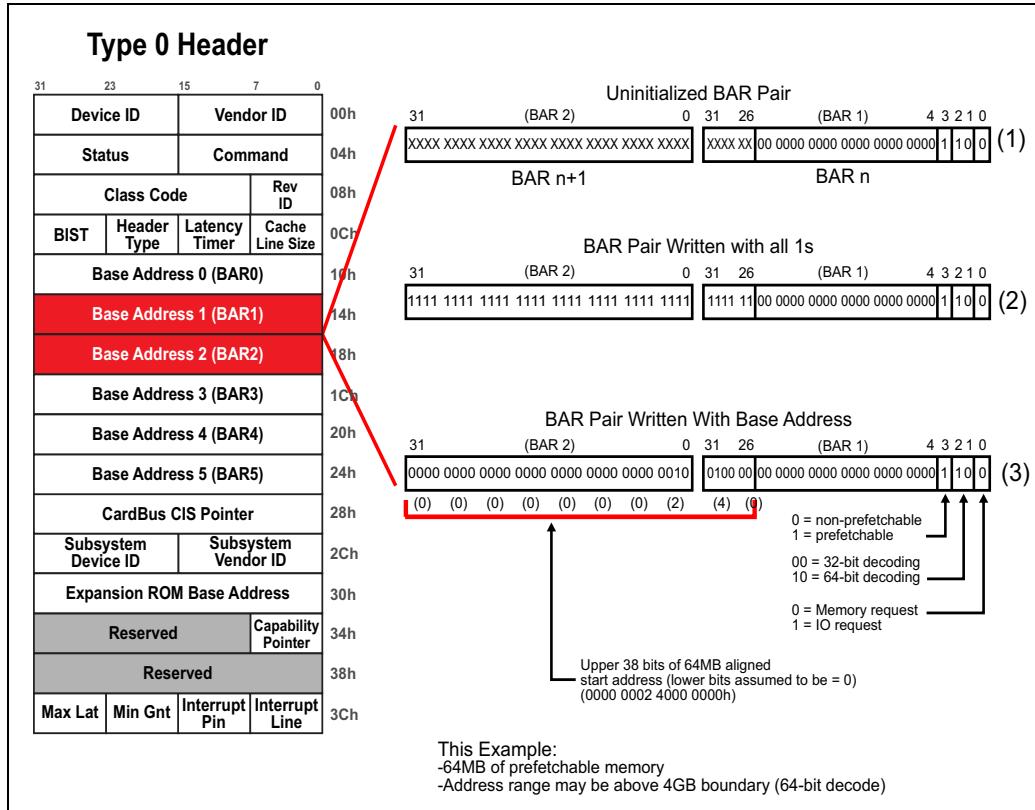


Table 4-2: Results Of Reading the BAR Pair after Writing All 1s To Both

BAR	BAR Bits	Meaning
Lower	0	Read as 0b, indicating a memory request. Since this is a memory request, bits 3:1 also have an encoded meaning.
Lower	2:1	Read as 10b indicating the target supports a 64-bit address decoder, and that the next sequential BAR contains the upper 32 bits of the address information.

## Chapter 4: Address Space & Transaction Routing

---

Table 4-2: Results Of Reading the BAR Pair after Writing All 1s To Both (Continued)

BAR	BAR Bits	Meaning
Lower	3	Read as 1b, indicating request is for prefetchable memory (meaning reads do not have side-effects); P-MMIO
Lower	25:4	Read as all 0s, indicating the size of the request (these bits are hard-coded to 0)
Lower	31:26	Read as all 1s because software has not yet programmed the upper bits with a start address for the block. Note that because bit 26 was the least significant writable bit, the memory address space request size is $2^{26}$ , or 64MB.
Upper	31:0	Read as all 1s. These bits will be used as the upper 32 bits of the 64-bit start address programmed by system software.

---

### BAR Example 3: IO Address Space Request

Continuing from the previous two examples, this same function is also requesting IO space, as shown in Figure 4-6 on page 134. In the diagram, the requesting BAR (BAR3 in the example) is shown at three points in the configuration process:

1. In (1) of Figure 4-6, we see the uninitialized state of the BAR. System software has previously written all 1s to every BAR and has evaluated BAR0, then BAR1 and BAR2. Now software is going to see if this device is requesting additional address space with BAR3. State (2) of Figure 4-6 shows the state of the BAR3 after the write of all 1s.
2. Software now reads in BAR3 to evaluate the size and type of the request. Table 4-3 on page 134 summarizes the results of this configuration read.
3. Now that software knows this is a request for 256 bytes of IO address space, the final step is to program the BAR with the base address of the IO address range being allocated to this device, specifically this BAR. State (3) of Figure 4-6 shows the state of the BAR after this step. In our example, the device start address is 16KB, so bit 14 is written resulting in a base address of 4000h; all other upper bits are cleared.

At this point, the configuration of BAR3 is complete. Once software enables IO address decoding in the Command register (offset 04h), the device will accept and respond to IO transactions within the range 4000h - 40FFh (256 bytes in size).

# PCI Express Technology

---

Figure 4-6: IO BAR Set Up

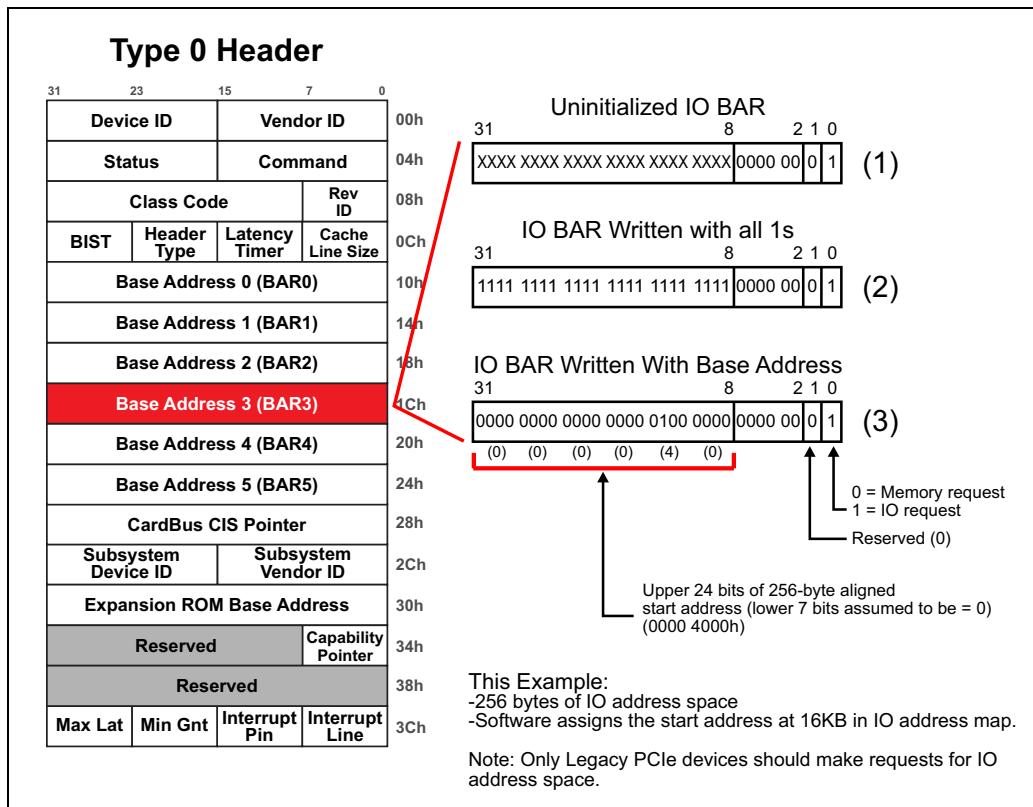


Table 4-3: Results Of Reading the IO BAR after Writing All 1s To It

BAR Bits	Meaning
0	Read as 1b, indicating an IO request. Since this is an IO request, bit 1 is reserved.
1	Reserved. Hard-coded to 0b.
7:2	Read as 0s Indicates size of the request (these bits are hard-coded to 0)
31:8	Read as 1s because software has not yet programmed the upper bits with a start address for the block. Note that because bit 8 was the least significant writable bit, the IO request size is $2^8$ , or 256 bytes.

### All BARs Must Be Evaluated Sequentially

After going through the previous three examples, it becomes clear that software must evaluate BARs in a sequential fashion.

Most of the time, functions do not need all six BARs. Even in the examples we went through, only four of the six available BARs were used. If the function in our example did not need to request any additional address space, the device designer would hard-code all bits of BAR4 and BAR5 to 0s. So even though software writes those BARs with all 1s, the writes have no affect. After evaluating BAR3, software would move on to evaluating BAR4. Once it detected that none of the bits were set, software would know this BAR is not being used and move on to evaluating the next BAR.

All BARs must be evaluated, even if software finds a BAR that is not being used. There are no rules in PCI or PCIe, that state that BAR0 must be the first BAR used for address space requests. If a device designer chooses to, they can use BAR4 for an address space request and hard-code BAR0, BAR1, BAR2, BAR3 and BAR5 to all 0s. This means software must evaluate every BAR in the header.

---

### Resizable BARs

The 2.1 version of the PCI Express specification added support for changing the size of the requested address space in the BARs by defining a new capability structure in extended config space. The new structure allows the function to advertise what address space sizes it can operate with and then have software enable one of the sizes based on the available system resources. For example, if a function would ideally like to have 2GB of prefetchable memory address space, but it could still operate with only 1GB, 512MB or 256MB of P-MMIO, system software may only enable the function to request 256MB of address space if software would not be able to accommodate a request of a larger size.

## Base and Limit Registers

---

### General

Once a function's BARs are programmed, the function knows what address range(s) it owns, which means that function will claim any transactions it sees that is targeting an address range it owns, an address range programmed into one of its BARs. This is good, but it's important to realize that the only way that function is going to "see" the transactions it should claim is if the bridge(s) upstream of it, forward those transactions downstream to the appropriate link that the target function is connected to. Therefore, each bridge (e.g. switch ports and root complex ports) needs to know what address ranges live beneath it so it can determine which requests should be forwarded from its primary interface (upstream side) to its secondary interface (downstream side). If the request is targeting an address that is owned by a BAR in a function beneath the bridge, the request should be forwarded to the bridge's secondary interface.

It is the Base and Limit registers in the Type 1 headers that are programmed with the range of addresses that live beneath this bridge. There are the three sets of Base and Limit registers found in each Type 1 header. Three sets of registers are needed because there can be three separate address ranges living below a bridge:

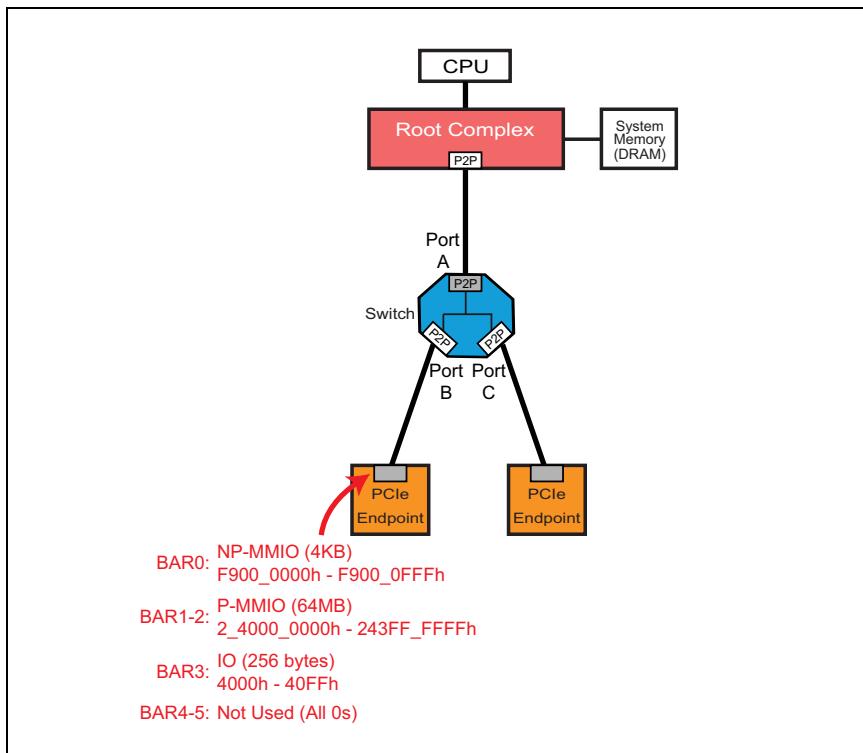
- Prefetchable Memory space (P-MMIO)
- Non-Prefetchable Memory space (NP-MMIO)
- IO space (IO)

To explain how these Base and Limit registers work, let's continue the example from the previous section and place that programmed function (an endpoint) beneath a switch as shown in Figure 4-7 on page 137. The figure also lists the address ranges owned by the BARs of that function.

The Base and Limit registers of every bridge upstream of the endpoint will need to be programmed, but to start out, we're going to focus on the bridge that is connected to the endpoint (Port B).

## Chapter 4: Address Space & Transaction Routing

Figure 4-7: Example Topology for Setting Up Base and Limit Values



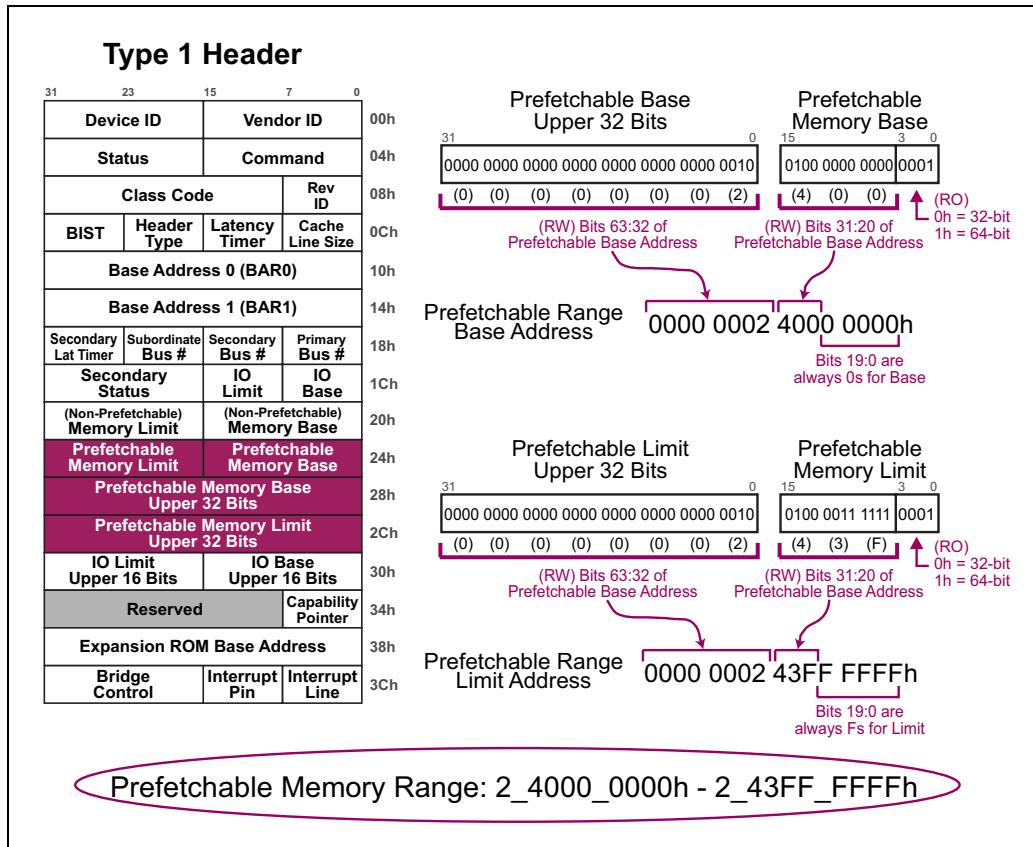
### Prefetchable Range (P-MMIO)

Type 1 headers have two pairs of prefetchable memory base/limit registers. The Prefetchable Memory Base/Limit registers store address info for the lower 32 bits of the prefetchable address range. If this bridge supports decoding 64-bit addresses, then the Prefetchable Memory Base/Limit Upper 32 Bits registers are also used and hold the upper 32 bits (bits [63:32]) of the address range. Figure 4-8 on page 138 shows the values software would program into these registers to indicate that the prefetchable address range of 2\_4000\_0000h - 2\_43FF\_FFFFh lives beneath that bridge (Port B). The meaning of each field in those registers is summarized in Table 4-4.

# PCI Express Technology

---

Figure 4-8: Example Prefetchable Memory Base/Limit Register Values



## Chapter 4: Address Space & Transaction Routing

---

Table 4-4: Example Prefetchable Memory Base/Limit Register Meanings

Register	Value	Use
Prefetchable Memory Base	4001h	The upper 12 bits of this register hold the upper 12 bits of the 32-bit BASE address (bits [31:20]). The lower 20 bits of the base address are implied to be all 0s, meaning the base address is always aligned on a 1MB boundary. The lower 4 bits of this register indicate whether a 64-bit address decoder is supported in the bridge, meaning the Upper Base/Limit Registers are used.
Prefetchable Memory Limit	43F1h	Similarly, the upper 12 bits of this register hold the upper 12 bits of the 32-bit LIMIT address (bits [31:20]). The lower 20 bits of the limit address are all implied to be all Fs. The lower 4 bits of this register have the same meaning as the lower 4 bits of the base register.
Prefetchable Memory Base Upper 32 Bits	00000002h	Holds the upper 32 bits of the 64-bit BASE address for Prefetchable Memory downstream of this port.
Prefetchable Memory Limit Upper 32 Bits	00000002h	Holds the upper 32 bits of the 64-bit LIMIT address for Prefetchable Memory downstream of this port.

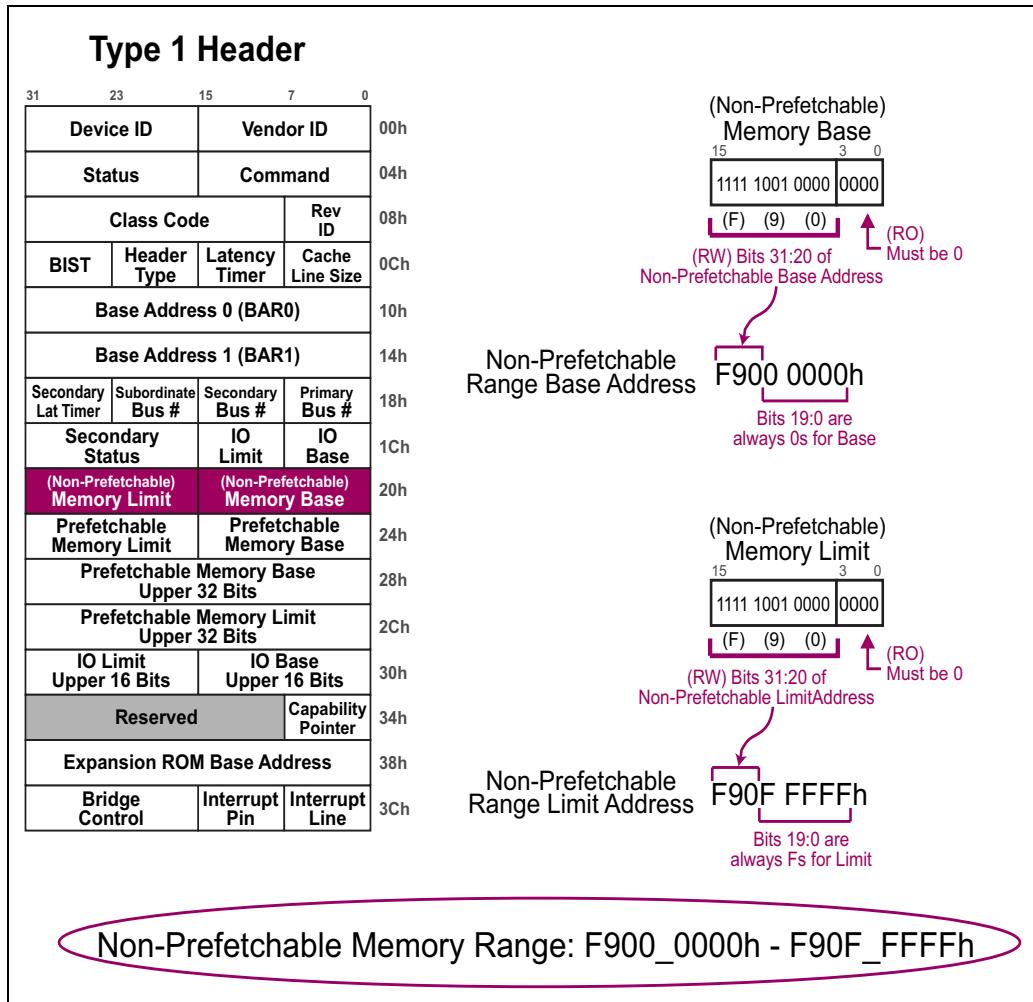
---

### Non-Prefetchable Range (NP-MMIO)

Unlike the prefetchable memory range, the non-prefetchable memory range can only support 32-bit addresses. So there is only one register for the base and one register for the limit. Following the example in Figure 4-7, the Non-Prefetchable Memory Base/Limit registers of Port B would be programmed with the values shown in Figure 4-9 on page 140. The meaning of these values is summarized in Table 4-5.

# PCI Express Technology

Figure 4-9: Example Non-Prefetchable Memory Base/Limit Register Values



## Chapter 4: Address Space & Transaction Routing

---

Table 4-5: Example Non-Prefetchable Memory Base/Limit Register Meanings

Register	Value	Use
(Non-Prefetchable) Memory Base	F900h	The upper 12 bits of this register hold the upper 12 bits of the 32-bit BASE address (bits [31:20]). The lower 20 bits of the base address are implied to be all 0s, meaning the base address is always aligned on a 1MB boundary. The lower 4 bits of this register must be 0s.
(Non-Prefetchable) Memory Limit	F900h	Similarly, the upper 12 bits of this register hold the upper 12 bits of the 32-bit LIMIT address (bits [31:20]). The lower 20 bits of the limit address are all implied to be all Fs. The lower 4 bits of this register must be 0s.

This example shows an interesting case where the non-prefetchable address range programmed in Port B's configuration space indicates a much larger range (1MB) than the NP-MMIO range (4KB) owned by the endpoint living downstream. This is because the memory base/limit registers in the Type 1 header, can only be used to specify address bits 20 and above ([31:20]). The lower 20 address bits, [19:0], are implied. So the smallest address range that can be specified with the memory base/limit registers is 1MB.

In our example, the endpoint requested, and was granted, 4KB of NP-MMIO (F900\_0000h - F900\_0FFFh). Port B was programmed with values indicating 1MB, or 1024KB, of NP-MMIO lived downstream of that port (F900\_0000h - F90F\_FFFFh). This means 1020KB (F900\_1000h - F90F\_FFFFh) of memory address space is wasted. This address space CANNOT be allocated to another endpoint because the routing of the packets would not work.

---

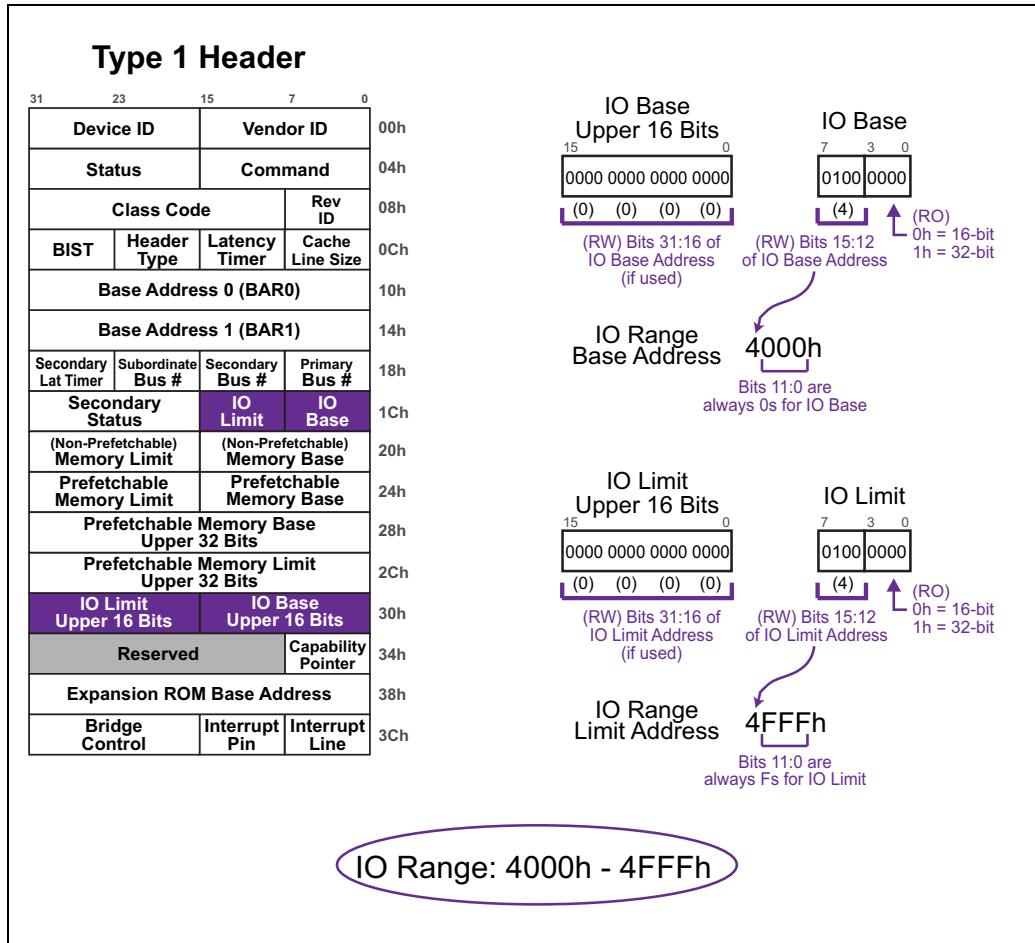
## IO Range

Like with the prefetchable memory range, Type 1 headers have two pairs of IO base/limit registers. The IO Base/Limit registers store address info for the lower 16 bits of the IO address range. If this bridge supports decoding 32-bit IO addresses (which is rare in real-world devices), then the IO Base/Limit Upper 16 Bits registers are also used and hold the upper 16 bits (bits [31:16]) of the IO

# PCI Express Technology

address range. Following our example, Figure 4-10 on page 142 shows the values software would program into these registers to indicate that the IO address range of 4000h - 4FFFh lives beneath that bridge (Port B). The meaning of each field in those registers is summarized in Table 4-6.

Figure 4-10: Example IO Base/Limit Register Values



## Chapter 4: Address Space & Transaction Routing

---

Table 4-6: Example IO Base/Limit Register Meanings

Register	Value	Use
IO Base	40h	The upper 4 bits of this register hold the upper 4 bits of the 16-bit BASE address (bits [15:12]). The lower 12 bits of the base address are implied to be all 0s, meaning the base address is always aligned on a 4KB boundary. The lower 4 bits of this register indicate whether a 32-bit IO address decoder is supported in the bridge, meaning the Upper Base/Limit Registers are used.
IO Limit	40h	Similarly, the upper 4 bits of this register hold the upper 4 bits of the 16-bit LIMIT address (bits [15:12]). The lower 12 bits of the limit address are all implied to be all Fs. The lower 4 bits of this register have the same meaning as the lower 4 bits of the base register.
IO Base Upper 16 Bits	0000h	Holds the upper 16 bits of the 32-bit BASE address for IO downstream of this port.
IO Limit Upper 16 Bits	0000h	Holds the upper 16 bits of the 32-bit LIMIT address for IO downstream of this port.

In this example, we see another situation where the address range programmed into the upstream bridge far exceeds the actual address range owned by the downstream function. The endpoint in our example owns 256 bytes of IO address space (specifically 4000h - 40FFh). Port B has been programmed with values indicating that 4KB of IO address space lives downstream (addresses 4000h - 4FFFh). Again, this is simply a limitation of Type 1 headers. For IO address space, the lower 12 bits (bits [11:0]) have implied values, so the smallest range of IO addresses that can be specified is 4KB. This limitation turns out to be more serious than the 1MB minimum window for memory ranges. In x86-based (Intel compatible) systems, the processors only support 16 bits of IO address space, and since only bits [15:12] of the IO address range can be specified in a bridge, that means that there can be a maximum of 16 ( $2^4$ ) different IO address ranges in a system.

## Unused Base and Limit Registers

Not every PCIe device will use all three types of address space. 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.

In the cases where an endpoint does not request all three types of address space, what are the base and limit registers of the bridges upstream of those devices programmed with? They can't be programmed with all 0s because the lower address bits would still be implied to be different (base = 0s; limit = Fs) which would represent a valid range. So to handle these cases, the limit register must be programmed with a higher address than the base. For example, if an endpoint does not request IO address space, then the bridge immediately upstream of that function would have its IO Base register programmed to 00h and its IO Limit register programmed with F0h. Since the limit address is higher than the base address, the bridge understands this is an invalid setting and takes it to mean that there are no functions downstream of it that own IO address space.

This method of invalidating base and limit registers is valid for all three base and limit pairs, not just for the IO base/limit registers.

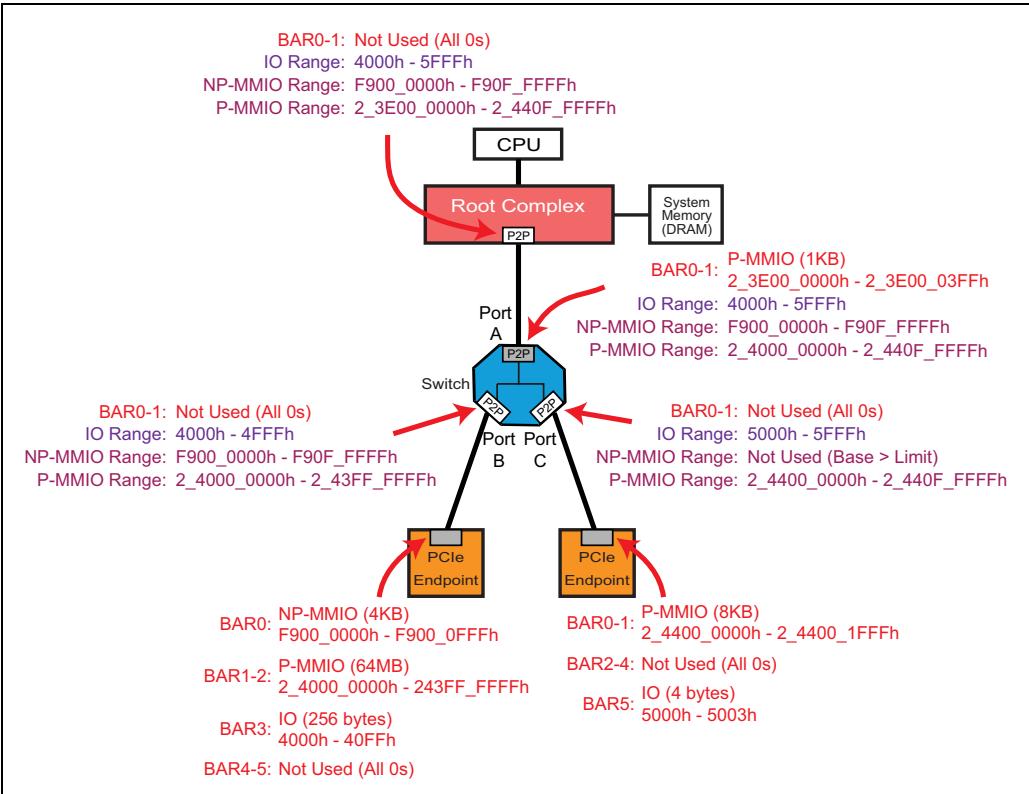
---

## Sanity Check: Registers Used For Address Routing

To ensure that you understand the rules and methods for setting up BARs and Base/Limit registers, please look over Figure 4-11 on page 145 to make sure it makes sense. We have simply extended the example system to include additional address space requests from the other endpoint, as well as from one of the switch ports (Port A). Remember that Type 1 headers also have BARs (two of them to be exact) and can request address space too. The Base/Limit registers in a bridge do NOT include the addresses owned by that same bridge's BARs. The Base/Limit registers only represent the addresses that live downstream of that bridge.

# Chapter 4: Address Space & Transaction Routing

Figure 4-11: Final Example Address Routing Setup



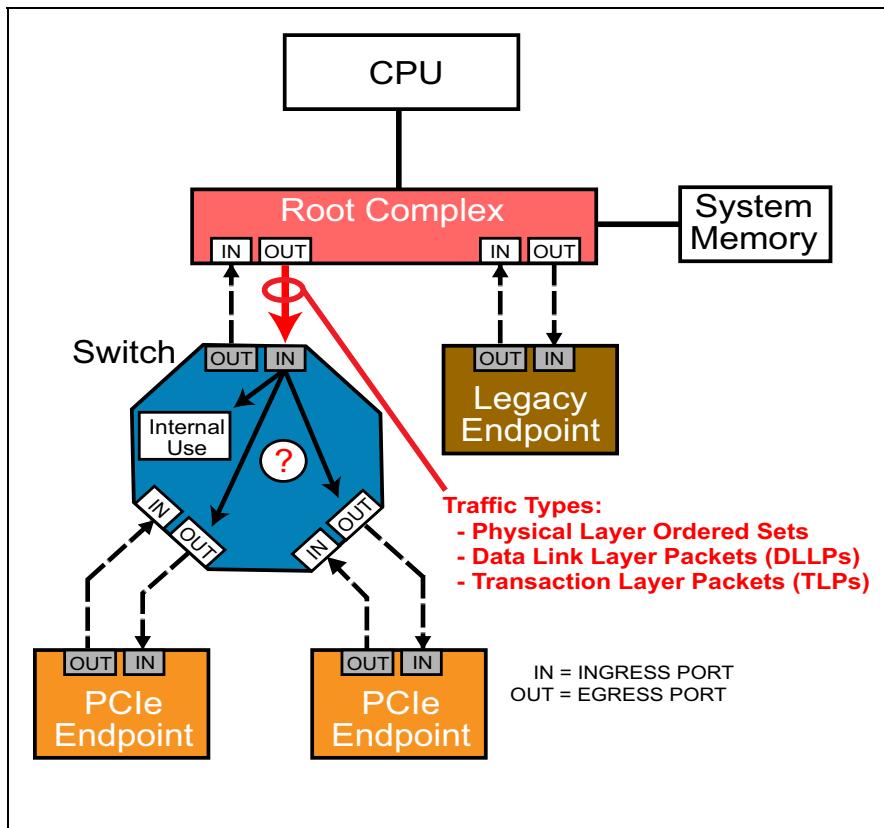
## TLP Routing Basics

The purpose of setting up the BARs and Base/Limit registers as described in the previous sections, is to ensure that traffic targeting a function will be routed correctly so the targeted function can see the transactions and claim them. In shared-bus architectures like PCI, all the traffic is visible to every device. The only time routing of requests happens is when the target is on another bus and must cross a bridge. Since PCIe Links are point-to-point, more routing will be needed to deliver transactions between devices.

# PCI Express Technology

---

Figure 4-12: Multi-Port PCIe Devices Have Routing Responsibilities



As illustrated in Figure 4-12 on page 146, a PCI Express topology consists of independent, point-to-point links connecting each device with one or more neighbors. As traffic arrives at the inbound side of a link interface (called the *ingress port*), the port checks for errors, then makes one of three decisions:

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

# **Chapter 4: Address Space & Transaction Routing**

---

## **Receivers Check For Three Types of Traffic**

Assuming a link is fully operational, the receiver interface of each device (ingress port) must detect and evaluate the arrival of the three types of link traffic: Ordered Sets, Data Link Layer Packets (DLLPs), and Transaction Layer Packets (TLPs). Ordered Sets and DLLPs are local to a link and thus are never routed to another link. TLPs can and do move from link to link, based on routing information contained in the packet headers.

---

## **Routing Elements**

Devices with multiple ports, like Root Complexes and Switches, can forward TLPs between the ports and are sometimes called Routing Agents or Routing Elements. They accept TLPs that target internal resources and forward TLPs between ingress and egress ports.

Interestingly, peer-to-peer routing support is required in Switches, but for a Root Complex it's optional. Peer-to-peer traffic is typically where one Endpoint sends packets that target another Endpoint.

Endpoints have only one Link and never expect to see ingress traffic other than what is targeting them. They simply accept or reject incoming TLPs.

---

## **Three Methods of TLP Routing**

### **General**

TLPs can be routed based on address (either memory or IO), based on ID (meaning Bus, Device, Function number), or routed implicitly. The routing method used is based on the TLP type. Table 4-7 on page 147 summarizes the TLP types and the routing methods used for each.

*Table 4-7: PCI Express TLP Types And Routing Methods*

<b>TLP Type</b>	<b>Routing Method Used</b>
Memory Read [Lock], Memory Write, AtomicOp	Address Routing
IO Read and Write	Address Routing

Table 4-7: PCI Express TLP Types And Routing Methods (Continued)

TLP Type	Routing Method Used
Configuration Read and Write	ID Routing
Message, Message With Data	Address Routing, ID Routing, or Implicit routing
Completion, Completion With Data	ID Routing

Messages are the only TLP type that support more than one routing method. Most of the message TLPs defined in the PCI Express spec use implicit routing, however, the vendor-defined messages could use address routing or ID routing if desired.

## Purpose of Implicit Routing and Messages

In implicit routing, neither address or ID routing information applies; instead, the packet is routed based on a code in the packet header indicating a destination with a known location in the topology, such as the Root Complex. This simplifies routing of messages in the cases where a type of implicit routing applies.

**Why Messages?** Message transactions were not defined in PCI or PCI-X, but were introduced with PCIe. The main reason for adding Messages as a packet type was to pursue the PCIe design goal to drastically reduce the number of sideband signals implemented in PCI (e.g. interrupt pins, error pins, power management signals, etc.). Consequently, most of the sideband signals were replaced with in-band packets in the form of Message TLPs.

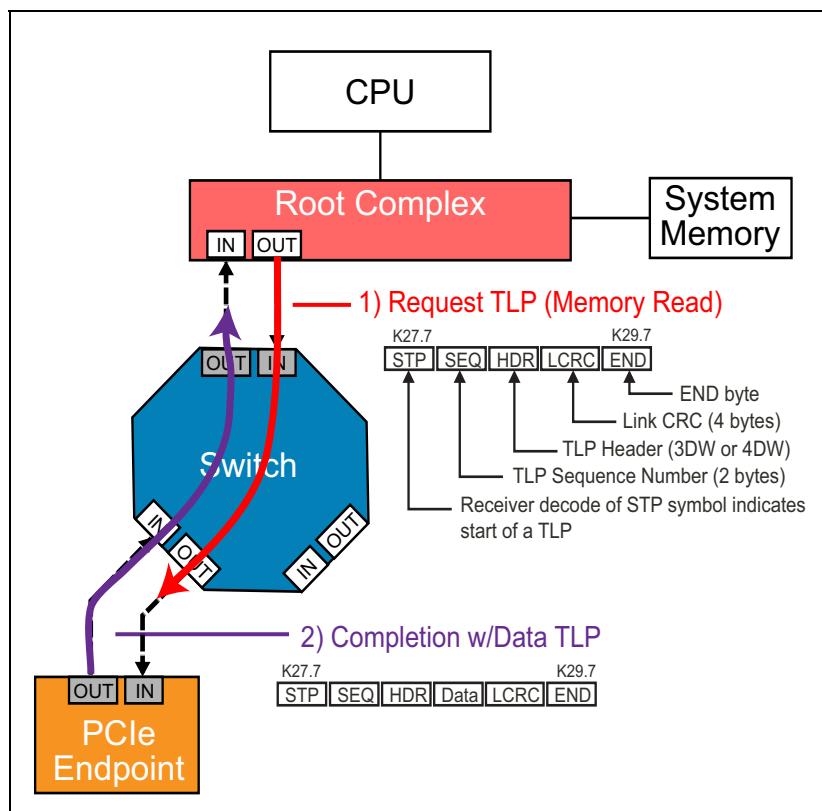
**How Implicit Routing Helps** Using in-band messages in place of sideband signals requires a means of routing them to the proper recipient in a topology consisting of numerous point-to-point links. Implicit routing takes advantage of the fact that Switches and other routing elements understand the concept of upstream and downstream, and that the Root Complex is found at the top of the topology while Endpoints are found at the bottom. As a result, a Message can use a simple code to show that it should go to the Root Complex, for example, or to be sent to all devices downstream. This ability eliminates the need to define address ranges or ID lists specifically used as the target of different message transactions.

The different types of implicit routing can be found in “Implicit Routing” on page 163.

### Split Transaction Protocol

Like most other serial technologies, PCI Express uses the split transaction protocol which allows a target device to receive one or more requests and then respond to each request with a separate completion. This is a significant improvement over the PCI bus protocol that used wait-states or delayed transactions (retries) to deal with latencies in accessing targets. Instead of testing to see when the target becomes ready to do a long-latency transfer, the target initiates the response whenever it's ready. This results in at least two separate TLPs per transaction - the Request and the Completion (as will be discussed later, a single read request may result in multiple completion TLPs being sent back). Figure 4-13 on page 149 illustrates the Request-Completion components of a split transaction. This example shows software reading data from an Endpoint.

Figure 4-13: PCI Express Transaction Request And Completion TLPs



## Posted versus Non-Posted

To mitigate the penalty of the Request-Completion latency, memory write transactions are posted, meaning the transaction is considered completed from the Requester's perspective as soon as the request leaves the Requester. If helpful, you can associate the term "posting" with the postal system, where posting a memory write is analogous to posting a letter in the mail. Once you've placed a letter in the postal box you put your faith in the system to deliver it and don't wait for verification of delivery. This approach can be much faster than waiting for the entire Request-Completion transit, but — as in all posting schemes — uncertainty exists concerning when (and if) the transaction completed successfully at the ultimate recipient.

In PCIe, the small amount of uncertainty involved by making all memory writes posted is considered acceptable in exchange for the performance gained. By contrast, writes to IO and configuration space almost always affect device behavior and have a timeliness associated with them. Consequently, it is important to know when (and if) those write requests completed. Because of this, IO writes and configuration writes are always non-posted and a completion will always be returned to report the status of the operation.

In summary, non-posted transactions require a completion. Posted transactions do not require, and should never receive, a completion. Table 4-8 on page 150 lists which PCIe transactions are posted and non-posted.

*Table 4-8: Posted and Non-Posted Transactions*

Request	How Request Is Handled
Memory Write	All <b>memory write requests are posted</b> . No completions are expected or sent.
Memory Read Memory Read Lock	All <b>memory read requests are non-posted</b> . A completion with data (made of one or more TLPs) will be returned by the Completer to deliver both the requested data and the status of the memory read. In the event of an error, a completion without data will be returned reporting the status.
AtomicOp	All <b>AtomicOp requests are non-posted</b> . A completion with data will be returned by the Completer containing the original value of the target location.

## Chapter 4: Address Space & Transaction Routing

---

Table 4-8: Posted and Non-Posted Transactions (Continued)

Request	How Request Is Handled
IO Read IO Write	All <b>IO requests are non-posted</b> . A completion without data will be returned for writes or failed reads, and a completion with data will be returned for successful reads.
Configuration Read Configuration Write	All <b>configuration requests are non-posted</b> . A completion without data will be returned for writes and failed reads, while a completion with data will be returned for successful reads.
Message	All <b>messages are posted</b> . The routing method depends on the Message type, but they're all considered posted requests.

---

## Header Fields Define Packet Format and Type

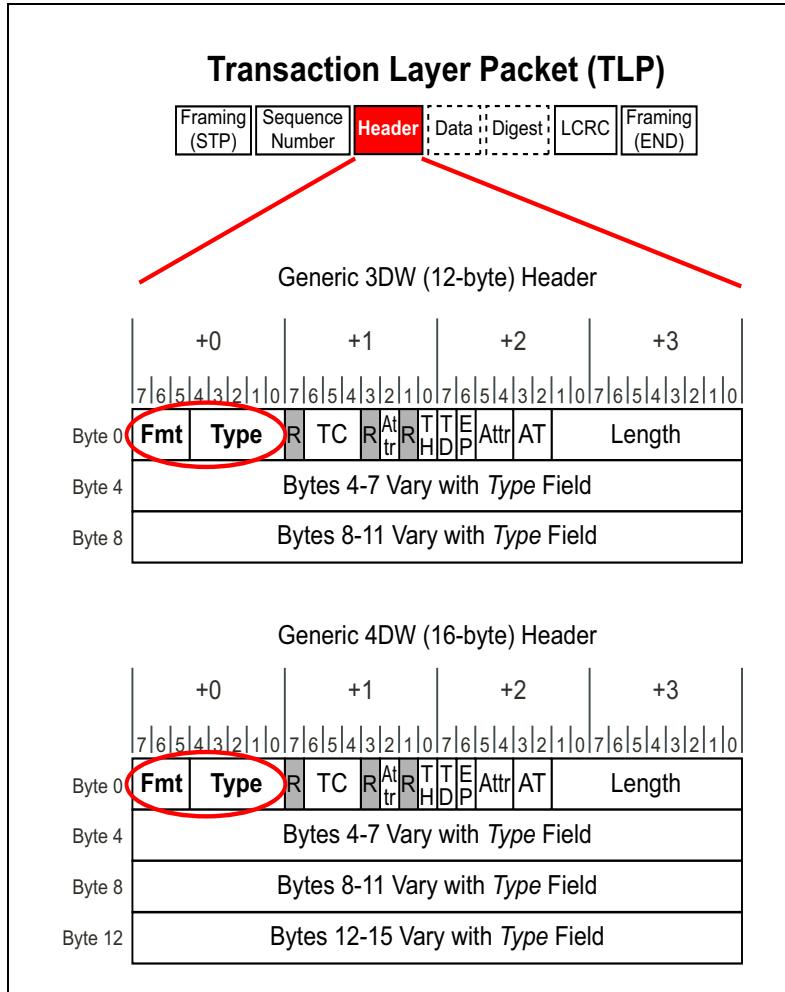
### General

As shown in Figure 4-14 on page 152, each TLP contains a three or four double-word (12 or 16 byte) header. This includes *Format* and *Type* fields that define the content of the rest of the header and indicate the routing method to be used for the TLP as it traverses the topology.

# PCI Express Technology

---

Figure 4-14: Transaction Layer Packet Generic 3DW And 4DW Headers



## Chapter 4: Address Space & Transaction Routing

---

### Header Format/Type Field Encodings

Table 4-9 on page 153 below summarizes the encodings used in TLP header Format and Type fields.

*Table 4-9: TLP Header Format and Type Field Encodings*

TLP	FMT[2:0]	TYPE [4:0]
Memory Read Request (MRd)	000 = 3DW, no data 001 = 4DW, no data	0 0000
Memory Read Lock Request (MRdLk)	000 = 3DW, no data 001 = 4DW, no data	0 0001
Memory Write Request (MWr)	010 = 3DW, w/ data 011 = 4DW, w/ data	0 0000
IO Read Request (IORd)	000 = 3DW, no data	00010
IO Write Request (IOWr)	010 = 3DW, w/ data	0 0010
Config Type 0 Read Request (CfgRd0)	000 = 3DW, no data	0 0100
Config Type 0 Write Request (CfgWr0)	010 = 3DW, w/ data	0 0100
Config Type 1 Read Request (CfgRd1)	000 = 3DW, no data	0 0101
Config Type 1 Write Request (CfgWr1)	010 = 3DW, w/ data	0 0101
Message Request (Msg)	001 = 4DW, no data	1 0RRR* (for RRR, see routing subfield in “Message Type Field Summary” on page 164)
Message Request w/Data (MsgD)	011 = 4DW, w/ data	1 0RRR* (for RRR, see routing subfield in “Message Type Field Summary” on page 164)

# PCI Express Technology

---

Table 4-9: TLP Header Format and Type Field Encodings (Continued)

TLP	FMT[2:0]	TYPE [4:0]
Completion (Cpl)	000 = 3DW, no data	0 1010
Completion W/Data (CplD)	010 = 3DW, w/ data	0 1010
Completion-Locked (CplLk)	000 = 3DW, no data	0 1011
Completion w/Data (CplDLk)	010 = 3DW, w/ data	0 1011
Fetch and Add AtomicOp Request (FetchAdd)	010 = 3DW, w/data 011 = 4DW, w/data	0 1100
Unconditional Swap AtomicOp Request (Swap)	010 = 3DW, w/data 011 = 4DW, w/data	0 1101
Compare and Swap AtomicOp Request (CAS)	010 = 3DW, w/data 011 = 4DW, w/data	0 1110
Local TLP Prefix (LPrfx)	100 = 1DW	0 LLLL
End-to-End TLP Prefix (EPrfx)	100 = 1DW	1 EEEE

---

## TLP Header Overview

When TLPs are received at an ingress port, they are first checked for errors at the Physical and Data Link Layers. If there are no errors, the TLP is examined at the Transaction Layer to learn which routing method is to be used. The basic steps are:

1. *Format* and *Type* fields determine the header size, format and type of the packet.
2. Depending on the routing method associated with the packet type, the device determines whether it's the intended recipient. If so, it will accept (consume) the TLP, but if not, it will forward the TLP to the appropriate egress port - subject to the rules for ordering and flow control for that egress port.
3. If this device is not the intended recipient nor is it in the path to the intended recipient, it will generally reject the packet as an Unsupported Request (UR).

## Applying Routing Mechanisms

Once the system addresses have been configured and transactions are enabled, devices examine incoming TLPs and use the corresponding configuration fields to route the packet. The following sections describe the basic features/functionality of each routing mechanism used in routing TLPs through the PCI Express fabric.

---

### ID Routing

ID routing is used to target the logical position - Bus Number, Device Number, Function Number (typically referred to as **BDF**), of a Function within the topology. It's compatible with routing methods used in the PCI and PCI-X protocols for configuration transactions. In PCIe, it is still used for routing configuration packets and is also used to route completions and some messages.

#### Bus Number, Device Number, Function Number Limits

PCI Express supports the same topology limits as PCI and PCI-X:

1. Eight bits are used to give the bus number, so a **maximum of 256 busses** are possible in a system. This includes internal busses created by Switches.
2. Five bits give the device number, so a **maximum of 32 devices** are possible per bus. An older PCI bus or an internal bus in a switch or root complex may host more than one downstream device. However, external PCIe links are always point-to-point and there's only one downstream device on the link. The device number for an external link is forced by the downstream port to always be Device 0, so every external Endpoint will always be Device 0 (unless using Alternative Routing-ID Interpretation (ARI), in which case, there are no device numbers; more about ARI can be found in the section on "IDO (ID-based Ordering)" on page 909).
3. Three bits give the function number, so a **maximum of 8 internal functions** is possible per device.

#### Key TLP Header Fields in ID Routing

If the Type field in a received TLP indicates ID routing is to be used, then the ID fields in the header (Bus, Device, Function) are used to perform the routing check. There are two cases: ID routing with a 3DW header and ID routing with a 4DW header (only possible in messages). Figure 4-15 on page 156 illustrates a TLP using ID routing and the 3DW header, while Figure 4-16 on page 156 shows the 4DW header for ID routing.

# PCI Express Technology

---

Figure 4-15: 3DW TLP Header - ID Routing Fields

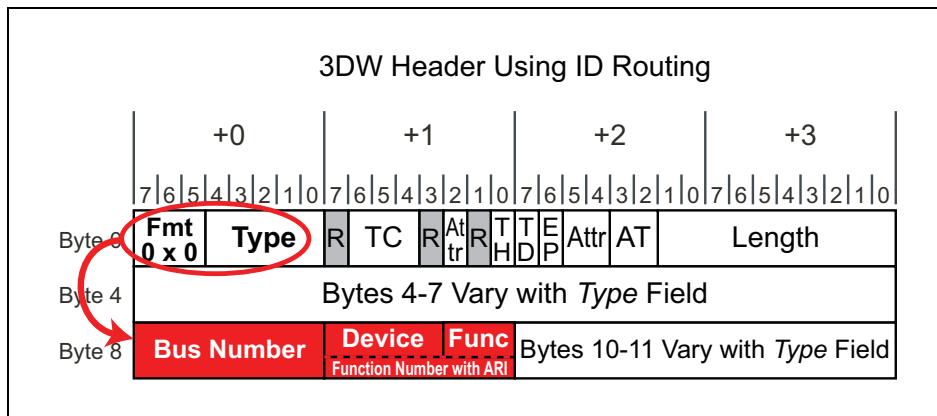
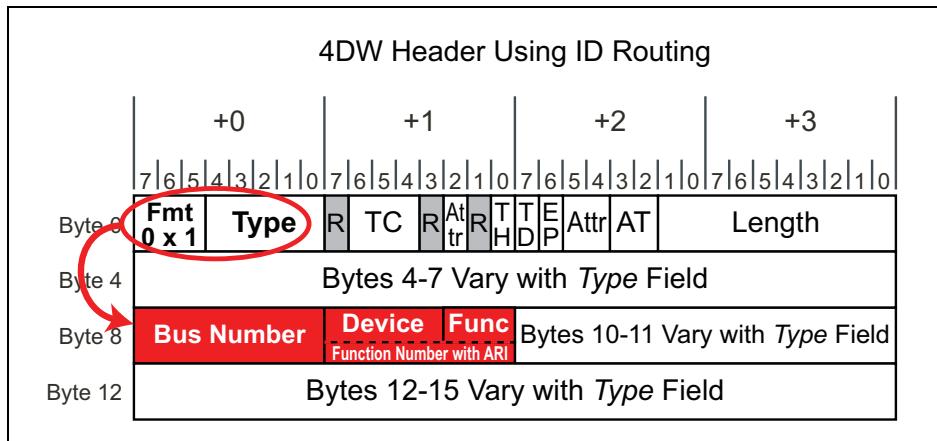


Figure 4-16: 4DW TLP Header - ID Routing Fields



## Endpoints: One Check

For ID routing, an Endpoint simply checks the ID field in the packet header against its own BDF. Each function “captures” its own Bus and Device Number every time a Type 0 configuration write is seen on its link from bytes 8-9 in the TLP Header. Where the captured Bus and Device Number information should be stored in not specified, only that functions must save it. The saved Bus and

## **Chapter 4: Address Space & Transaction Routing**

---

Device numbers are used as the Requester ID in TLP requests that this Endpoint initiates so the Completer of that request can include the Requester ID value in the completion packet(s). The Requester ID in a completion packet is used to route the completion.

### **Switches (Bridges): Two Checks Per Port**

For an ID-routed TLP, a switch port first checks to see whether it is the intended target by comparing the target ID in the TLP Header against its own BDF, as shown by (1) in Figure 4-17 on page 158. As was true for an Endpoint, each switch port captures its own Bus and Device number every time a configuration write (Type 0) is detected on its Upstream Port. If the target ID field in the TLP agrees with the ID of the switch port, it consumes the packet. If the ID field doesn't match, it then checks to see if the TLP is targeting a device below this switch port. It does this by checking the Secondary and Subordinate Bus Number registers to see if the target Bus Number in the TLP is within this range (inclusive). If so, then the TLP should be forwarded downstream. This check is indicated by (2) in Figure 4-17 on page 158. If the packet was moving downstream (arrived on the Upstream Port) and doesn't match the BDF of the Upstream Port or fall within the Secondary-Subordinate bus range, it will be handled as an Unsupported Request on the Upstream Port.

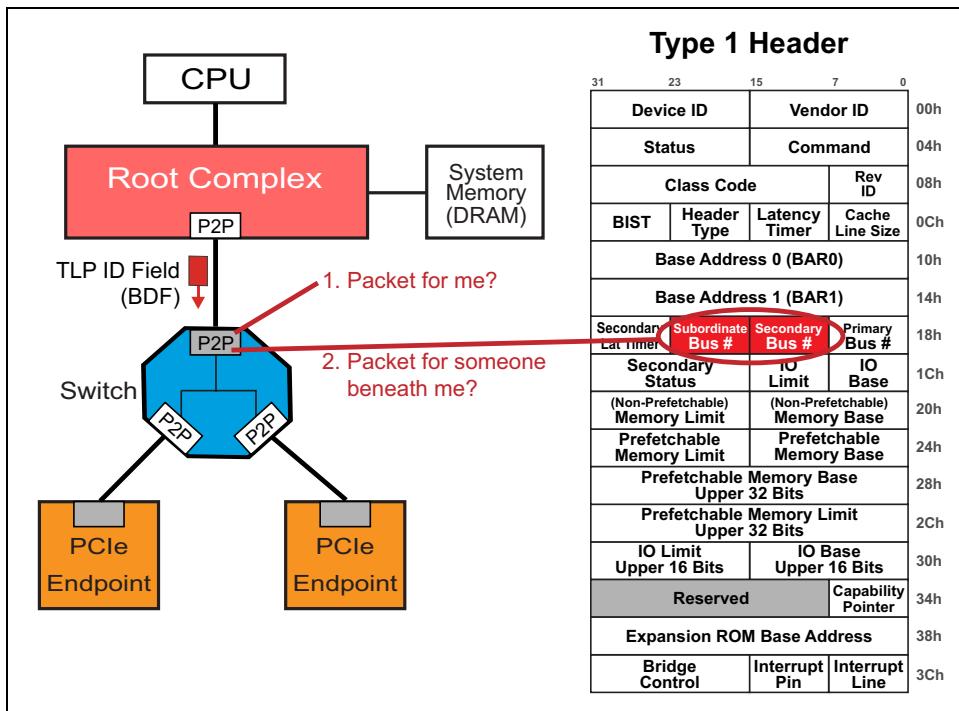
If the Upstream Port determines that a TLP it received is for one of the devices beneath it (because the target bus number was within the range of its Secondary-Subordinate bus number range), then it forwards it downstream and all the downstream ports of the switch perform the same checks. Each downstream port checks to see if the TLP is targeting them. If so, the targeted port will consume the TLP and the other ports ignore it. If not, all downstream ports check to see if the TLP is targeting a device beneath their port. The one port that returns true on that check will forward the TLP to its Secondary Bus and the other downstream ports ignore the TLP.

In this section, it is important to remember that each port on a switch is a Bridge, and thus has its own configuration space with a Type 1 Header. Even though Figure 4-17 on page 158 only shows a single Type 1 Header, in reality, each port (each P2P Bridge) has its own Type 1 Header and performs the same two checks on TLPs when they are seen by that port.

# PCI Express Technology

---

Figure 4-17: Switch Checks Routing Of An Inbound TLP Using ID Routing



---

## Address Routing

TLPs that use address routing refer to the same memory (system memory and memory-mapped IO) and IO address maps that PCI and PCI-X transactions do. Memory requests targeting an address below 4GB (i.e. a 32-bit address) must use a 3DW header, and requests targeting an address above 4GB (i.e. a 64-bit address) must use a 4DW header. IO requests are restricted to 32-bit addresses and are only implemented to support legacy functionality.

# Chapter 4: Address Space & Transaction Routing

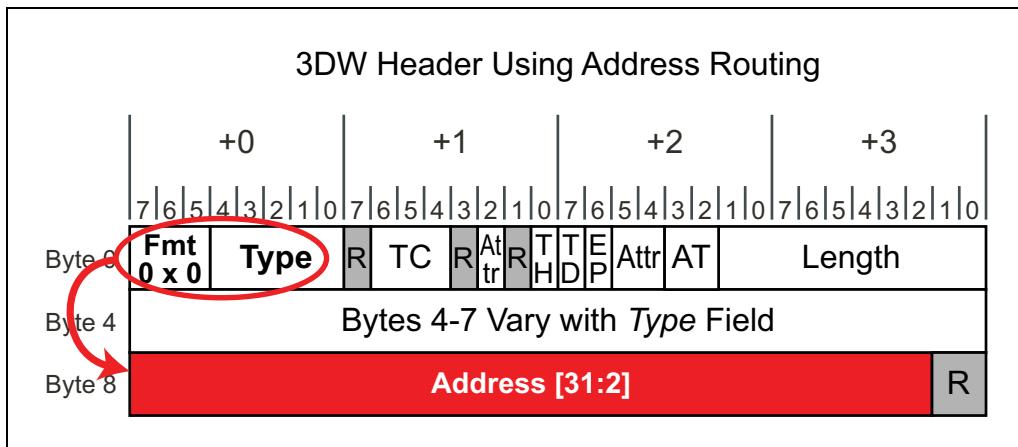
## Key TLP Header Fields in Address Routing

When the Type field indicates address routing is to be used for a TLP, then the Address Fields in the header are used to perform the routing check. These can be 32-bit addresses or 64-bit addresses.

**TLPs with 32-Bit Address** For IO or 32-bit memory requests, a 3DW header is used as shown in Figure 4-18. The memory-mapped registers targeted with these TLPs will therefore reside below the 4GB memory or IO address boundary.

**TLPs with 64-Bit Address** For 64-bit memory requests, a 4DW header is used as shown in Figure 4-19 on page 160. The memory-mapped registers targeted with these TLPs are able to reside above the 4GB memory boundary.

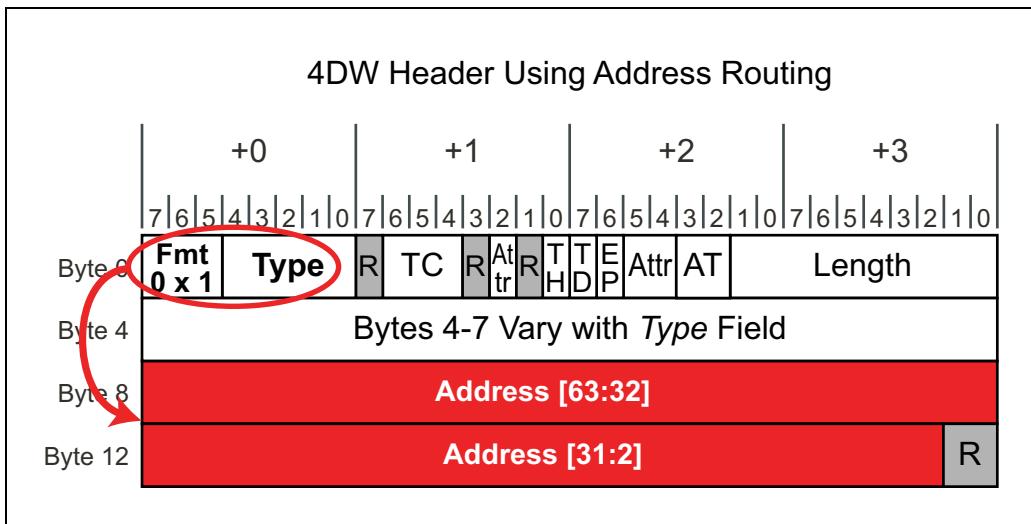
Figure 4-18: 3DW TLP Header - Address Routing Fields



# PCI Express Technology

---

Figure 4-19: 4DW TLP Header - Address Routing Fields

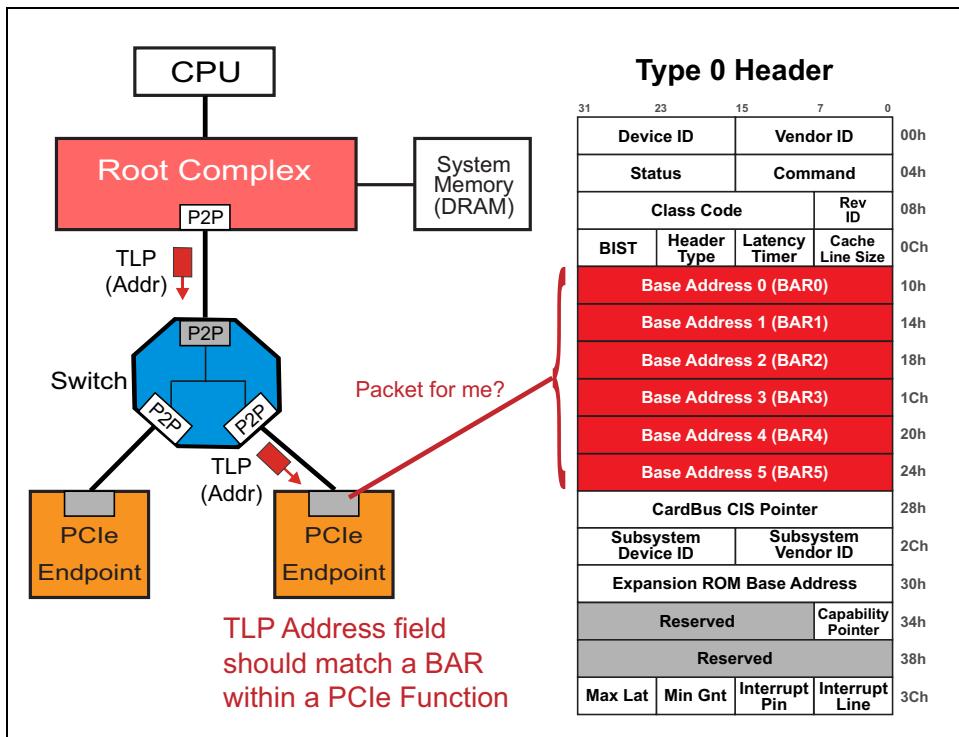


## Endpoint Address Checking

If an Endpoint receives a TLP that uses address routing then it checks the address in the header against each of its implemented Base Address Registers (BARs) in its configuration header, as shown in Figure 4-20. Since Endpoints only have one link interface, it will either accept the packet or reject it. The Endpoint will accept the packet if the target address in the TLP matches one of the ranges programmed into its BARs. More info on how the BARs are used can be found in section “Base Address Registers (BARs)” on page 126.

## Chapter 4: Address Space & Transaction Routing

Figure 4-20: Endpoint Checks Incoming TLP Address

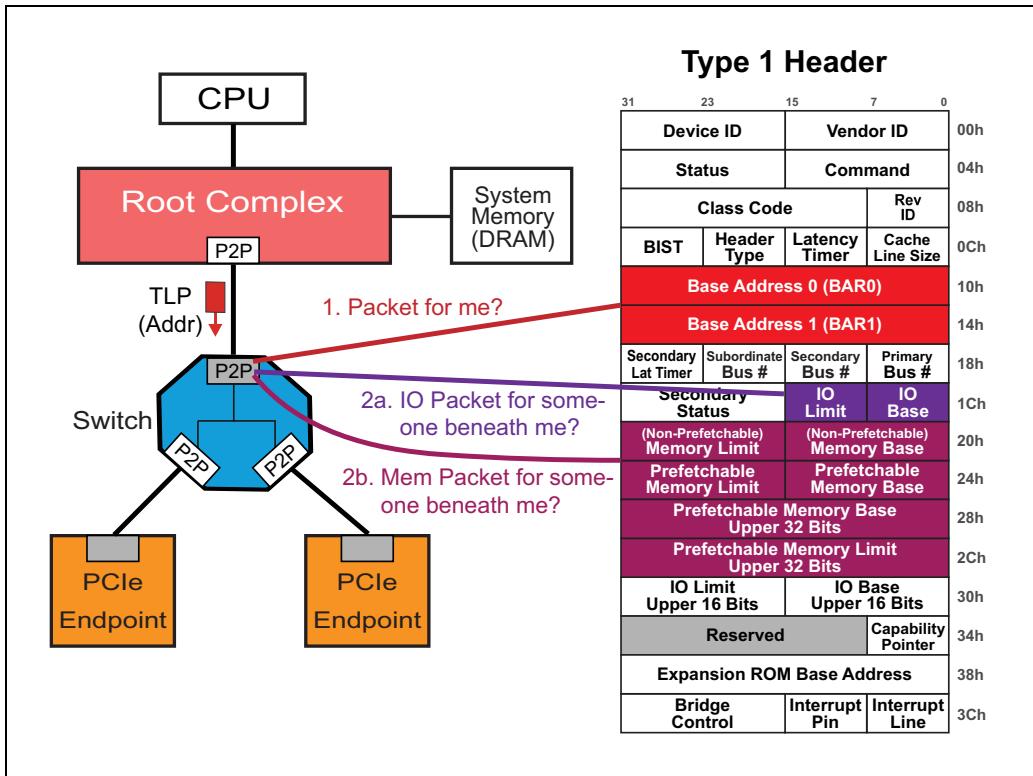


### Switch Routing

If an incoming TLP uses address routing, a Switch Port first checks to see if the address is local within the Port itself by comparing the address in the packet header against its two BARs in its Type 1 configuration header, as shown in Step 1 of Figure 4-21 on page 162. If it matches one of these BARs, the switch port is the target of the TLP and consumes the packet. If not, the port then checks its Base/Limit register pairs to see if the TLP is targeting a function beneath (downstream of) this bridge. If the Request targets IO space, it will check the IO Base and Limit registers, as shown in Step 2a. However, if the Request targets memory space, it will check the Non-prefetchable Memory Base/ Limit registers and the Prefetchable Memory Base/Limit registers, as indicated by Step 2b in Figure 4-21 on page 162. More info on how the Base/Limit register pairs are evaluated can be found in section “Base and Limit Registers” on page 136.

# PCI Express Technology

Figure 4-21: Switch Checks Routing Of An Inbound TLP Using Address



To understand routing of address-based TLPs in switches, it is good to remember that each switch port is its own bridge. Below are the steps that a bridge (switch port) takes upon receiving an address-based TLP:

## Downstream Traveling TLPs (Received on Primary Interface)

1. IF the target address in the TLP matches one of the BARs, then this bridge (switch port) consumes the TLP because it is the target of the TLP.
2. IF the target address in the TLP falls in the range of one of its Base/Limit register sets, the packet will be forwarded to the secondary interface (downstream).
3. ELSE the TLP will be handled as an Unsupported Request on the primary interface. (This is true if no other bridges on the primary interface claim the TLP either.)

# **Chapter 4: Address Space & Transaction Routing**

---

## **Upstream Traveling TLPs (Received on Secondary Interface)**

1. IF the target address in the TLP matches one of the BARs, then this bridge (switch port) consumes the TLP because it is the target of the TLP.
2. IF the target address in the TLP falls in the range of one of its Base/Limit register sets, the TLP will be handled as an Unsupported Request on the secondary interface. (This is true unless this port is the upstream port of the switch. In these cases, the packet may be a peer-to-peer transaction and will be forwarded downstream on a different downstream port than the one it was received on.)
3. ELSE the TLP will be forwarded to the primary interface (upstream) given that the TLP address is not for this bridge and is not for any function beneath this bridge.

## **Multicast Capabilities**

The 2.1 version of the PCI Express specification added support for specifying a range of addresses that provide multicast functionality. Any packets received that fall within the address range specified as the multicast range are routed/accepted according to the multicast rules. This address range might not be reserved in a function's BARs and might not be within a bridge's Base/Limit register pair, but would still need to be accepted/forwarded appropriately. More info can be found on the multicast functionality in the section on "Multicast Capability Registers" on page 889.

---

## **Implicit Routing**

Implicit routing, used in some message packets, is based on the awareness of routing elements that the topology has upstream and downstream directions and a single Root Complex at the top. This allows some simple routing methods without the need to assign a target address or ID. Since the Root Complex generally integrates power management, interrupt, and error handling logic, it is either the source or recipient of most PCI Express messages.

## **Only for Messages**

Some messages use address or ID routing rather than implicit routing, and for them, the routing mechanisms are applied in the same way as described in the those sections. However, most messages use implicit routing. The purpose of implicit routing is to mimic side-band signal behavior since a design goal for PCIe was to eliminate as many side-band signals from PCI as possible. These

# PCI Express Technology

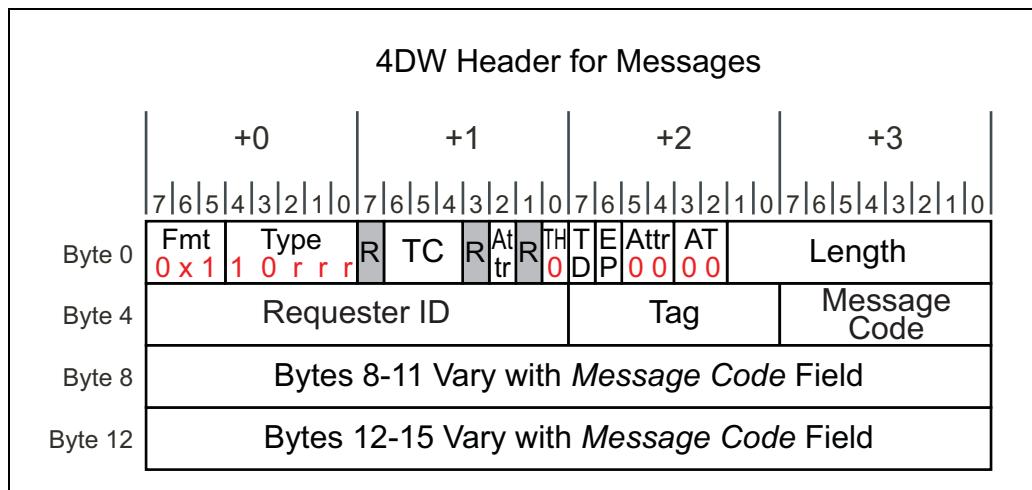
side-band signals in PCI were typically either the host notifying all devices of an event or devices notifying the host of an event. In PCIe, we have Message TLPs to convey these events. The types of events that PCIe has defined messages for are:

- Power Management
- INTx legacy interrupt signaling
- Error signaling
- Locked Transaction support
- Hot Plug signaling
- Vendor-specific signaling
- Slot Power Limit settings

## Key TLP Header Fields in Implicit Routing

For implicit routing, the routing sub-field in the header is used to determine the message destination. Figure 4-22 on page 164 illustrates a message TLP using implicit routing.

Figure 4-22: 4DW Message TLP Header - Implicit Routing Fields



## Message Type Field Summary

Table 4-10 on page 165 shows how the TLP header Type field for Messages is interpreted. As shown, the upper two bits indicate the packet is a Message while the lower three bits specify the routing method to apply. Note that Message TLPs always use a 4DW header regardless of the routing option selected.

## Chapter 4: Address Space & Transaction Routing

---

For address routing, bytes 8-15 contain up to a 64-bit address, and for ID routing, bytes 8 and 9 contain the target BDF.

*Table 4-10: Message Request Header Type Field Usage*

Type Field Bits	Description
Bit 4:3	Defines the type of transaction: 10b = Message TLP
Bit 2:0	Message Routing Subfield R[2:0] <ul style="list-style-type: none"><li>• 000b = Implicit - Route to the Root Complex</li><li>• 001b = Route by Address (bytes 8-15 of header contain address)</li><li>• 010b = Route by ID (bytes 8-9 of header contain ID)</li><li>• 011b = Implicit - Broadcast downstream</li><li>• 100b = Implicit - Local: terminate at receiver</li><li>• 101b = Implicit - Gather &amp; route to the Root Complex</li><li>• 110b - 111b = Reserved: terminate at receiver</li></ul>

### Endpoint Handling

For implicit routing, an Endpoint simply checks whether the routing sub-field is appropriate for it. For example, an Endpoint will accept a Broadcast Message or a Message that terminates at the receiver; but not Messages that implicitly target the Root Complex.

### Switch Handling

Routing elements like Switches consider the port on which the TLP arrived on and whether the routing sub-field code is appropriate for it. For example:

1. A Switch Upstream Port may legitimately receive a Broadcast Message. It will duplicate that and forward it to all its Downstream Ports. An implicitly routed Broadcast Message received on a Downstream Port of a Switch (meaning the message was traveling upstream) would be an error that would be handled as a Malformed TLP.
2. A Switch may receive implicitly routed Messages for the Root Complex on Downstream Ports and will forward these to its Upstream Port because the location of the Root Complex is understood to be upstream. It would not accept Messages received on its Upstream Port (meaning the message was traveling downstream) that are implicitly routed to the Root Complex.

# **PCI Express Technology**

---

3. If an implicitly routed Message indicates it should terminate at the receiver, then the receiving switch port will consume the message rather than forward it.
4. For messages routed using address or ID routing, a Switch will simply perform normal address or ID checks in deciding whether to accept or forward it.

---

## **DLLPs and Ordered Sets Are Not Routed**

DLLP and Ordered Set traffic is not routed from ingress ports to egress ports of switches or root complexes. These packets move from port to port across a link from Physical Layer to Physical Layer.

DLLPs originate at the Data Link Layer of a PCI Express port, pass through the Physical Layer, exit the port, traverse the Link and arrive at the neighboring port. At this port, the packet passes through the Physical Layer and ends up at the Data Link Layer where it is processed and consumed. DLLPs do not proceed further up the port to the Transaction Layer and hence are not routed.

Similarly, Ordered-Set packets originate at the Physical Layer, exit the port, traverse the Link and arrive at the neighboring port. At this port, the packet arrives at the Physical Layer where it is processed and consumed. Ordered-Sets do not proceed further up the port to the Data Link Layer and Transaction Layer and hence are not routed.

As has been discussed in this chapter, only TLPs are routed through switches and root complexes. They originate at the Transaction Layer of a source port and end up at the Transaction Layer of a destination port.

# 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

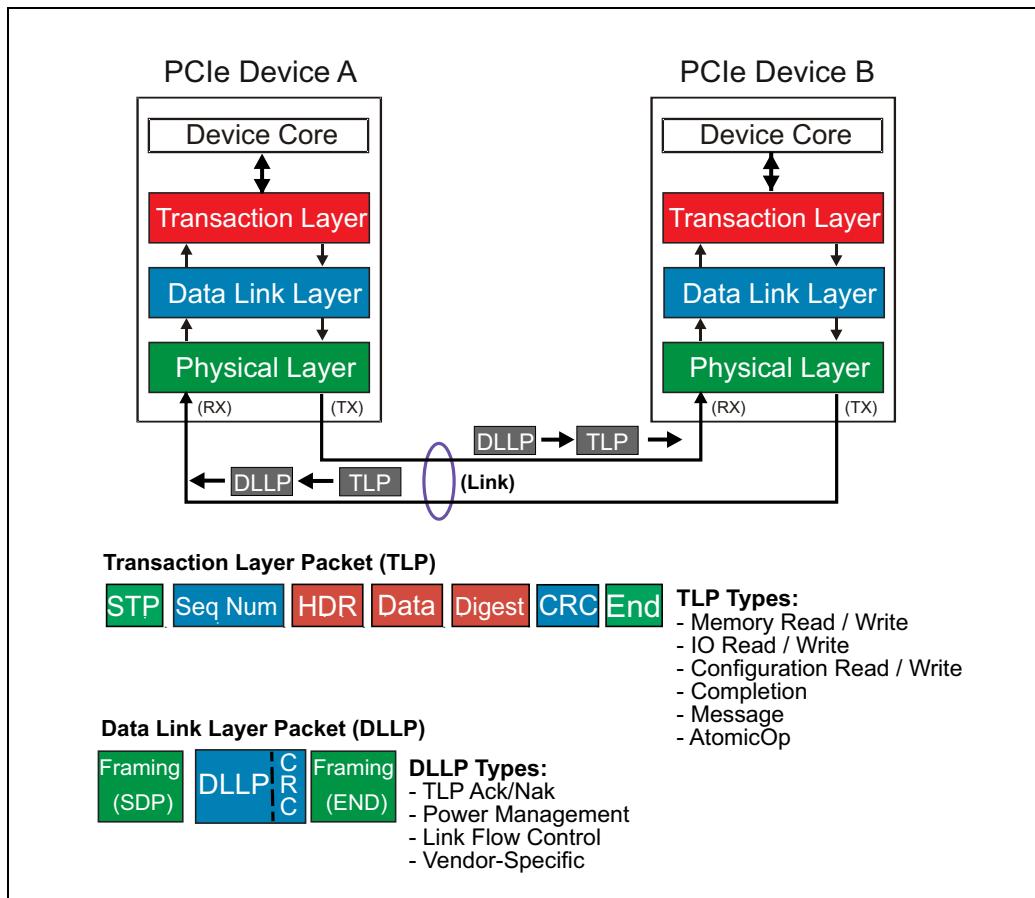
Unlike parallel buses, serial transport buses like PCIe use no control signals to identify what's happening on the Link at a given time. Instead, the bit stream they send must have an expected size and a recognizable format to make it pos-

# PCI Express Technology

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.

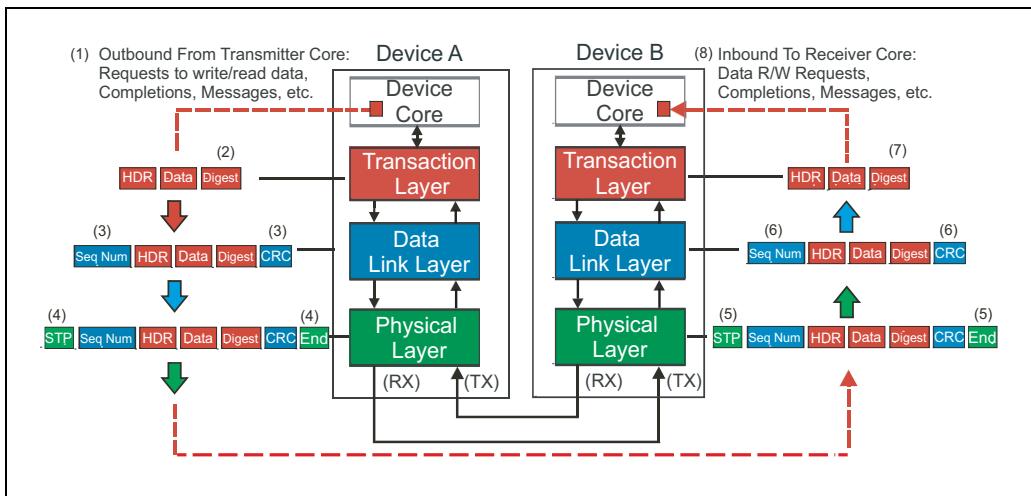
#### Transmitter:

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.)

# Chapter 5: TLP Elements

2. Based on that request, the Transaction Layer builds the TLP header, appends any data payload, and optionally calculates and appends the digest (End-to-End CRC, ECRC) if that's supported and has been enabled. At this point the TLP is placed into a Virtual Channel buffer. The Virtual Channel manages the sequence of TLPs according to the Transaction Ordering rules and also verifies that the receiver has enough flow control credits to accept a TLP before it can be passed down to the Data Link Layer.
3. When it arrives at the Data Link Layer, the TLP is assigned a Sequence Number and then a Link CRC is calculated based on the contents of the TLP and that Sequence Number. A copy of the resulting packet is saved in the Retry Buffer in case of transmission errors while it is also passed on to the Physical Layer.

Figure 5-2: PCIe TLP Assembly/Disassembly



4. The Physical Layer does several things to prepare the packet for serial transmission, including byte striping, scrambling, encoding, and serializing the bits. For Gen1 and Gen2 devices, when using 8b/10b encoding, the control characters STP and END are added to either end of the packet. Finally, the packet is transmitted across the Link. In Gen3 mode, STP token is added to the front end of a TLP, but END is not added to the end of the packet. Rather the STP token contains information about TLP packet size.

## Receiver:

5. At the Receiver (Device B in this example), everything done to prepare the packet for transmission must now be undone. The Physical Layer de-serializes the bit stream, decodes the resulting symbols, and un-stripes the bytes.

# PCI Express Technology

---

The control characters are removed here because they only have meaning at the Physical Layer, and then the packet is forwarded to the Data Link Layer.

6. The Data Link Layer calculates the CRC and compares it to the received CRC. If that matches, the Sequence Number is checked. If there are no errors, the CRC and Sequence Number are removed and the TLP is passed to the Transaction Layer of the receiver and notifies the sender of good reception by returning an Ack DLLP. In the event of an error a Nak will be returned instead, and the transmitter will re-replay TLPs in its Retry Buffer.
7. At the Transaction Layer, the TLP is decoded and the information is passed to the core logic for appropriate action. If the receiving device is the final target of this packet, it checks for ECRC errors and reports any related ECRC error condition to the core logic should there be any.

---

## TLP Structure

The basic usage of each field in a Transaction Layer Packet is defined in Table 5-1 on page 174.

*Table 5-1: TLP Header Type Field Defines Transaction Variant*

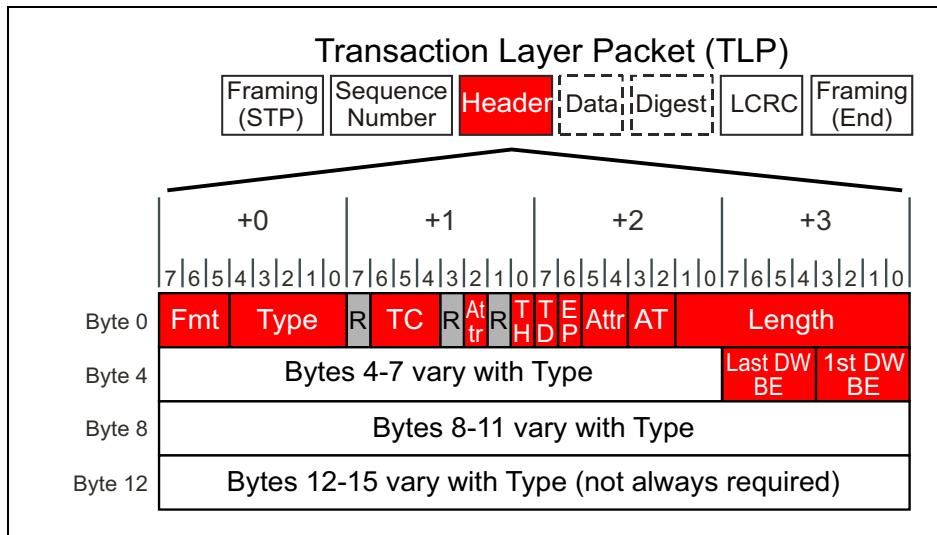
TLP Component	Protocol Layer	Component Use
Header	Transaction Layer	3 or 4DW (12 or 16 bytes) in size. Format varies with type, but Header defines parameters, including: <ul style="list-style-type: none"><li>• Transaction type</li><li>• Target address, ID, etc.</li><li>• Transfer size (if any), Byte Enables</li><li>• Attributes</li><li>• Traffic Class</li></ul>
Data	Transaction Layer	Optional 1-1024 DW Payload, which is qualified with Byte Enables or byte-aligned start and end addresses. Note that a length of zero can't be specified, but a zero-length read (useful in some cases) can be approximated by specifying a length of 1 DW and Byte Enables of all zero. The resulting data from the Completer will be undefined but the Requester doesn't use it, so the result is the same.
Digest/ECRC	Transaction Layer	Optional. When present, ECRC is always 1 DW in size.

## Generic TLP Header Format

### General

Figure 5-3 on page 175 illustrates the format and contents of a generic TLP 4DW header. In this section, fields common to nearly all transactions are summarized. Header format differences associated with specific transaction types are covered later.

Figure 5-3: Generic TLP Header Fields



### Generic Header Field Summary

Table 5-2 on page 176 summarizes the size and use of each of the generic TLP header fields. Note that fields marked “R” in Figure 5-3 on page 175 are reserved and should be set to zero.

# PCI Express Technology

---

Table 5-2: Generic Header Field Summary

Header Field	Header Location	Field Use
Fmt[2:0] (Format)	Byte 0 Bit 7:5	<p>These bits encode information about header size and whether a data payload will be part of the TLP:</p> <ul style="list-style-type: none"><li>00b 3DW header, no data</li><li>01b 4DW header, no data</li><li>10b 3DW header, with data</li><li>11b 4DW header, with data</li></ul> <p>An address below 4GB must use a 3DW header. The spec states that receiver behavior is undefined if 4DW header is used for an address below 4GB with the upper 32 bits of the 64-bit address set to zero.</p>
Type[4:0]	Byte 0 Bit 4:0	<p>These bits encode the transaction variant used with this TLP. The Type field is used with Fmt [1:0] field to specify transaction type, header size, and whether data payload is present. See “Generic Header Field Details” on page 178 for details.</p>
TC [2:0] (Traffic Class)	Byte 1 Bit 6:4	<p>These bits encode the traffic class to be applied to this TLP and to the completion associated with it (if any):</p> <ul style="list-style-type: none"><li>000b = Traffic Class 0 (Default)</li><li>.</li><li>.</li><li>111b = Traffic Class 7</li></ul> <p>TC 0 is the default class, while TC 1-7 are used to provide differentiated services. See “Traffic Class (TC)” on page 247 for additional information.</p>
Attr [2] (Attributes)	Byte 1 Bit 2	<p>This third Attribute bit indicates whether ID-based Ordering is to be used for this TLP. To learn more, see “ID Based Ordering (IDO)” on page 301.</p>
TH (TLP Processing Hints)	Byte 1 Bit 0	<p>Indicates when TLP Hints have been included to give the system some idea about how best to handle this TLP. See “TPH (TLP Processing Hints)” on page 899 for a discussion on their usage.</p>

## Chapter 5: TLP Elements

---

*Table 5-2: Generic Header Field Summary (Continued)*

Header Field	Header Location	Field Use
TD (TLP Digest)	Byte 2 Bit 7	<p>If TD = 1, the optional 4-byte TLP Digest has been included with this TLP as the ECRC value.</p> <p><u>Some rules:</u></p> <ul style="list-style-type: none"> <li>• Presence of the Digest field must be checked by all receivers based on this bit.</li> <li>• A TLP with TD = 1 but no Digest is handled as a Malformed TLP.</li> <li>• If a device supports checking ECRC and TD=1, it must perform the ECRC check.</li> <li>• If a device does not support checking ECRC (optional) at the ultimate destination, it must ignore the digest.</li> </ul> <p>For more on this topic see “CRC” on page 653 and “ECRC Generation and Checking” on page 657.</p>
EP (Poisoned Data)	Byte 2 Bit 6	<p>If EP = 1, the data accompanying this data should be considered invalid although the transaction is being allowed to complete normally. For more on poisoned packets, refer to “Data Poisoning” on page 660.</p>
Attr [1:0] (Attributes)	Byte 2 Bit 5:4	<p><u>Bit 5 = Relaxed ordering:</u> When set to 1, PCI-X relaxed ordering is enabled for this TLP. If 0, then strict PCI ordering is used.</p> <p><u>Bit 4 = No Snoop:</u> When set to 1, Requester is indicating that no host cache coherency issues exist for this TLP. System hardware can thus save time by skipping the normal processor cache snoop for this request. When 0, PCI -type cache snoop protection is required.</p>

# PCI Express Technology

---

Table 5-2: Generic Header Field Summary (Continued)

Header Field	Header Location	Field Use
Address Type [1:0]	Byte 2 Bit 3:2	For Memory and Atomic Requests, this field supports address translation for virtualized systems. The translation protocol is described in a separate spec called <i>Address Translation Services</i> , where it can be seen that the field encodes as: 00 = Default/Untranslated 01 = Translation Request 10 = Translated 11 = Reserved
Length [9:0]	Byte 2 Bit 1:0 Byte 3 Bit 7:0	TLP data payload transfer size, in DW. Encoding: 00 0000 0001b = 1DW 00 0000 0010b = 2DW . . . 11 1111 1111b = 1023 DW 00 0000 0000b = 1024 DW
Last DW Byte Enables [3:0]	Byte 7 Bit 7:4	These four high-true bits map one-to-one to the bytes within the last double word of payload. Bit 7 = 1: Byte 3 in last DW is valid; otherwise not Bit 6 = 1: Byte 2 in last DW is valid; otherwise not Bit 5 = 1: Byte 1 in last DW is valid; otherwise not Bit 4 = 1: Byte 0 in last DW is valid; otherwise not
First DW Byte Enables [3:0]	Byte 7 Bit 3:0	These four high-true bits map one-to-one to the bytes within the first double word of payload. Bit 3 = 1: Byte 3 in first DW is valid; otherwise not Bit 2 = 1: Byte 2 in first DW is valid; otherwise not Bit 1 = 1: Byte 1 in first DW is valid; otherwise not Bit 0 = 1: Byte 0 in first DW is valid; otherwise not

---

## Generic Header Field Details

In the following sections, we describe details of each TLP Header field depicted in Figure 5-3 on page 175.

# Chapter 5: TLP Elements

---

## Header Type/Format Field Encodings

Table 5-3 on page 179 summarizes the encodings used in TLP header Type and Format (Fmt) fields.

*Table 5-3: TLP Header Type and Format Field Encodings*

TLP	FMT[2:0]	TYPE [4:0]
Memory Read Request (MRd)	000 = 3DW, no data 001 = 4DW, no data	0 0000
Memory Read Lock Request (MRdLk)	000 = 3DW, no data 001 = 4DW, no data	0 0001
Memory Write Request (MWr)	010 = 3DW, w/ data 011 = 4DW, w/ data	0 0000
IO Read Request (IORd)	000 = 3DW, no data	0 0010
IO Write Request (IOWr)	010 = 3DW, w/ data	0 0010
Config Type 0 Read Request (CfgRd0)	000 = 3DW, no data	0 0100
Config Type 0 Write Request (CfgWr0)	010 = 3DW, w/ data	0 0100
Config Type 1 Read Request (CfgRd1)	000 = 3DW, no data	0 0101
Config Type 1 Write Request (CfgWr1)	010 = 3DW, w/ data	0 0101
Message Request (Msg)	001 = 4DW, no data	1 0 rrr* (see routing field)
Message Request W/Data (MsgD)	011 = 4DW, w/ data	1 0 rrr* (see routing field)
Completion (Cpl)	000 = 3DW, no data	0 1010
Completion W/Data (CplD)	010 = 3DW, w/ data	0 1010
Completion-Locked (CplLk)	000 = 3DW, no data	0 1011
Completion W/Data (CplDLk)	010 = 3DW, w/ data	0 1011
Fetch and Add AtomicOp Request	010 = 3DW, w/ data 011 = 4DW, w/ data	0 1100

# PCI Express Technology

---

Table 5-3: TLP Header Type and Format Field Encodings (Continued)

TLP	FMT[2:0]	TYPE [4:0]
Unconditional Swap AtomicOp Request	010 = 3DW, w/ data 011 = 4DW, w/ data	0 1101
Compare and Swap AtomicOp Request	010 = 3DW, w/ data 011 = 4DW, w/ data	0 1110
Local TLP Prefix	100 = TLP Prefix	0L <sub>3</sub> L <sub>2</sub> L <sub>1</sub> L <sub>0</sub>
End-to-End TLP Prefix	100 = TLP Prefix	1E <sub>3</sub> E <sub>2</sub> E <sub>1</sub> E <sub>0</sub>

## Digest / ECRC Field

The TLP Digest bit reports the presence of the End-to-End CRC (ECRC). If this optional feature is supported and enabled by software, devices calculate and apply an ECRC for all TLPs they originate. Note that using ECRC requires devices to include the optional Advanced Error Reporting registers, since the capability and control registers for it are located there.

**ECRC Generation and Checking.** ECRC covers all fields that do not change as the TLP is forwarded across the fabric. However, there are two bits that can legally change as a packet makes its way across a topology:

- **Bit 0 of the Type field** — changes when a configuration transaction is forwarded across a bridge and changes from a type 1 to a type 0 configuration transaction because it has reached the targeted bus. This is accomplished by changing bit 0 of the type field.
- **Error/Poisoned (EP) bit** — this can change as a TLP traverses the fabric if the data associated with the packet is seen as corrupted. This is an optional feature referred to as error forwarding.

**Who Checks ECRC?** The intended target of an ECRC is the ultimate recipient of the TLP. Checking the LCRC verifies no transmission errors across a given Link, but that gets recalculated for the packet at the egress port of a routing element (Switch or Root Complex) before being forwarded to the next Link, which could mask an internal error in the routing element. To protect against that, the ECRC is carried forward unchanged on its journey between the Requester and Completer. When the target device checks the ECRC, any error possibilities along the way have a high probability of being detected.

The spec makes two statements regarding a Switch's role in ECRC checking:

- A Switch that supports ECRC checking performs this check on TLPs destined to a location within the Switch itself. "On all other TLPs a Switch must preserve the ECRC (forward it untouched) as an integral part of the TLP."
- "Note that a Switch may perform ECRC checking on TLPs passing through the Switch. ECRC Errors detected by the Switch are reported in the same way any other device would report them, but do not alter the TLPs passage through the Switch."

## Using Byte Enables

**General.** Like PCI, PCIe needs a mechanism to reconcile its DW-aligned addresses with the need, at times, for transfer sizes or starting/ending addresses that are not DW aligned. Toward this end, PCI Express makes use of the two Byte Enable fields introduced earlier in Figure 5-3 on page 175 and in Table 5-2 on page 176. The First DW Byte Enable field and the Last DW Byte Enable fields allow the Requester to qualify the bytes of interest within the first and last double words transferred.

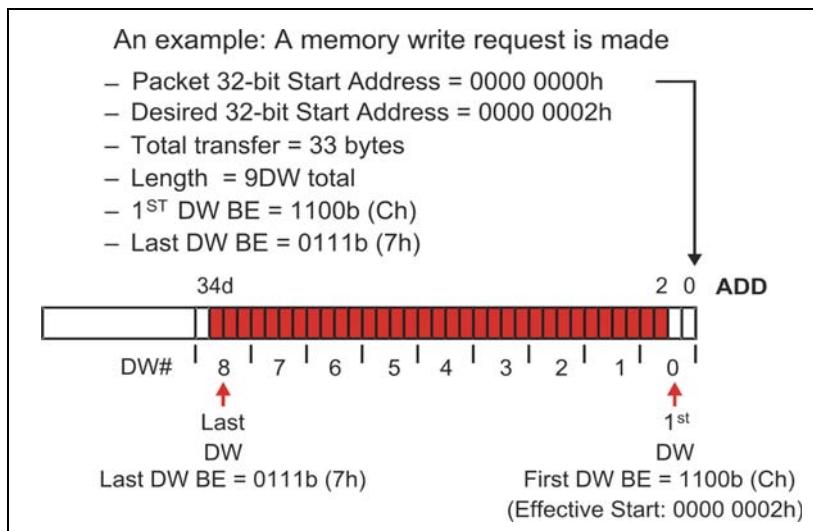
## Byte Enable Rules

1. Byte enable bits are high true. A value of 0 indicates the corresponding byte in the data payload should not be used by the Completer. A value of 1 indicates it should.
2. If the valid data is all within a single double word, the Last DW Byte enable field must be = 0000b.
3. If the header Length field indicates a transfer is more than 1DW, the First DW Byte Enable must have at least one bit enabled.
4. If the Length field indicates a transfer of 3DW or more, then the First DW Byte Enable field and the Last DW Byte Enable field must have contiguous bits set. In these cases, the Byte Enables are only being used to give the byte offset of the effective starting and ending address from the DW-aligned address.
5. Discontinuous byte enable bit patterns in the First DW Byte enable field are allowed if the transfer is 1DW.
6. Discontinuous byte enable bit patterns in both the First and Second DW Byte enable fields are allowed if the transfer is between one and two DWS.
7. A write request with a transfer length of 1DW and no byte enables set is legal, but has no effect on the Completer.
8. If a read request of 1 DW has no byte enables set, the completer returns a 1DW data payload of undefined data. This may be used as a Flush mechanism that takes advantage of transaction ordering rules to force all previously posted writes out to memory before the completion is returned.

# PCI Express Technology

**Byte Enable Example.** An example of byte enable use in this case is illustrated in Figure 5-4 on page 182. Note that the transfer length must extend from the first DW with any valid byte enabled to the last DW with any valid bytes enabled. Because the transfer is more than 2DW, the byte enables may only be used to specify the start address location (2d) and end address location (3d) of the transfer.

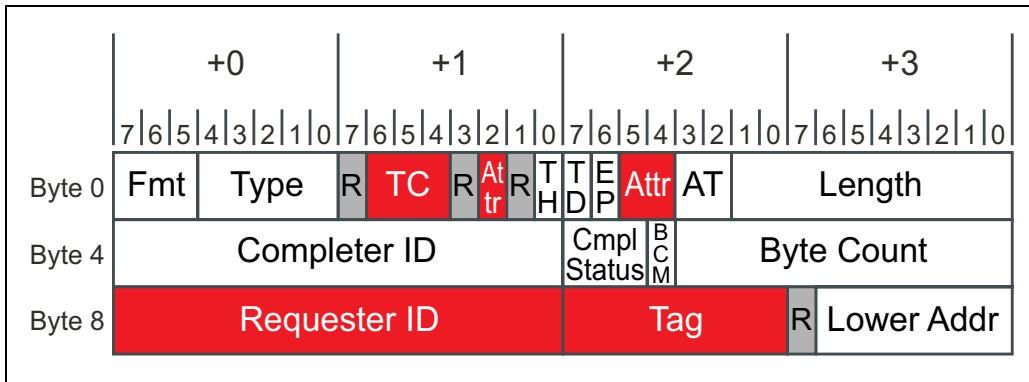
*Figure 5-4: Using First DW and Last DW Byte Enable Fields*



## Transaction Descriptor Fields

As transactions move between requester and completer, it's necessary to uniquely identify a transaction, since many split transactions may be queued up from the same Requester at any instant. To help with this, the spec defines several important header fields that form a unique Transaction Descriptor, as illustrated in Figure 5-5.

*Figure 5-5: Transaction Descriptor Fields*



While the Transaction Descriptor fields are not in adjacent header locations, collectively they describe key transaction attributes, including:

**Transaction ID.** The combination of the Requester ID (Bus, Device, and Function Number of the Requester) and the Tag field of the TLP.

**Traffic Class.** The Traffic Class (TC) is added by the requester based on the core logic request and travels unmodified through the topology to the Completer. On every Link, the TC is mapped to one of the Virtual Channels.

**Transaction Attributes.** The ID-based Ordering, Relaxed Ordering, and No Snoop bits also travel with the Request packet to the Completer.

## Additional Rules For TLPs With Data Payloads

The following rules apply when a TLP includes a data payload.

1. The Length field refers only to the data payload.
2. The first byte of data in the payload (immediately after the header) is always associated with the lowest (start) address.
3. The Length field always represents an integral number of DWs transferred. Partial DWs are qualified using First and Last Byte Enable fields.
4. The spec states that, when multiple transactions are returned by a completer in response to a single memory request, each intermediate transaction must end on naturally-aligned 64- or 128-byte address boundaries for a Root Complex. This is controlled by a configuration bit called the Read Completion Boundary (RCB). All other devices follow the PCI-X protocol

and break such transactions at naturally-aligned 128-byte boundaries. This makes buffer management simpler in bridges.

5. The Length field is reserved when sending Message Requests unless the message is the version with data (*MsgD*).
6. The TLP data payload must not exceed the current value in the Max\_Payload\_Size field of the Device Control Register. Only write transactions have data payloads, so this restriction doesn't apply to read requests. A receiver is required to check for violations of the Max\_Payload\_Size limit during writes, and violations are treated as Malformed TLPs.
7. Receivers also must check for discrepancies between the value in the Length field and the actual amount of data transferred in a TLP. This type of violation is also treated as a Malformed TLP.
8. Requests must not mix combinations of start address and transfer length that would cause a memory access to cross a 4KB boundary. While checking for this is optional, if seen it's treated as a Malformed TLP.

---

## Specific TLP Formats: Request & Completion TLPs

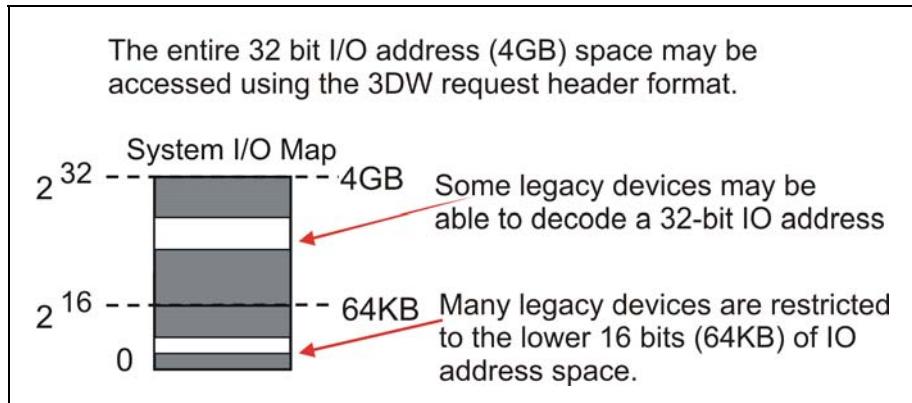
In this section, the format of 3DW and 4DW headers used to accomplish specific transaction types are described. Many of the generic fields described previously apply, but an emphasis is placed on the fields which are handled differently with specific transaction types. Detailed description of TLP Header format are described in sections following for TLP types: 1) IO Request, 2) Memory Requests, 3) Configuration Requests, 4) Completions and 5) Message Requests.

### IO Requests

While the spec discourages the use of IO transactions, allowance is made for Legacy devices and for software that may need to rely on a compatible device residing in the system IO map rather than the memory map. While the IO transactions can technically access a 32-bit IO range, in reality many systems (and CPUs) restrict IO access to the lower 16 bits (64KB) of this range. Figure 5-6 on page 185 depicts the system IO map and the 16- and 32-bit address boundaries. Devices that don't identify themselves as Legacy devices are not permitted to request IO address space in their configuration Base Address Registers.

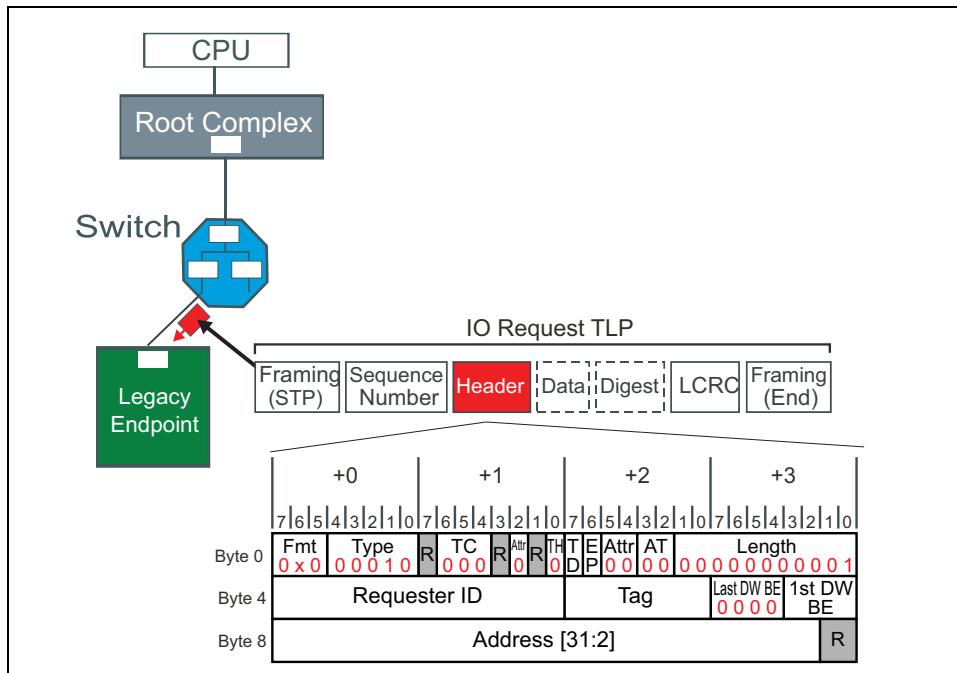
## Chapter 5: TLP Elements

Figure 5-6: System IO Map



**IO Request Header Format.** A 3 DW IO request header is shown in Figure 5-7 on page 185 and each of the fields is described in the section that follows.

Figure 5-7: 3DW IO Request Header Format



# PCI Express Technology

---

**IO Request Header Fields.** The location and use of each field in an IO request header is described in Table 5-4 on page 186.

*Table 5-4: IO Request Header Fields*

Field Name	Header Byte/Bit	Function
Fmt [2:0] (Format)	Byte 0 Bit 7:5	Packet Format for IO requests: 000b = IO Read (3DW without data) 010b = IO Write (3DW with data)
Type [4:0]	Byte 0 Bit 4:0	Packet type is 00010b for IO requests
TC [2:0] (Traffic Class)	Byte 1 Bit 6:4	Traffic Class for IO requests is always zero, ensuring that these packets will never interfere with any high-priority packets.
Attr [2] (Attributes)	Byte 1 Bit 2	ID-based Ordering doesn't apply for IO requests and this bit is reserved.
TH (TLP Processing Hints)	Byte 1 Bit 0	TLP processing Hints don't apply to IO requests and this bit is reserved.
TD (TLP Digest)	Byte 2 Bit 7	Indicates the presence of a digest field (ECRC) at the end of the TLP.
EP (Poisoned Data)	Byte 2 Bit 6	Indicates whether the data payload (if present) is poisoned.
Attr [1:0] (Attributes)	Byte 2 Bit 5:4	Relaxed Ordering and No Snoop bits don't apply for IO requests and are always zero.
AT [1:0] (Address Type)	Byte 2 Bit 3:2	Address Type doesn't apply for IO requests and these bits must be zero.
Length [9:0]	Byte 2 Bit 1:0 Byte 3 Bit 7:0	Indicates data payload size in DW. For IO requests, this field is always just 1 since no more than 4 bytes can be transferred. The First DW Byte Enables qualify which bytes are used.

## Chapter 5: TLP Elements

---

Table 5-4: IO Request Header Fields (Continued)

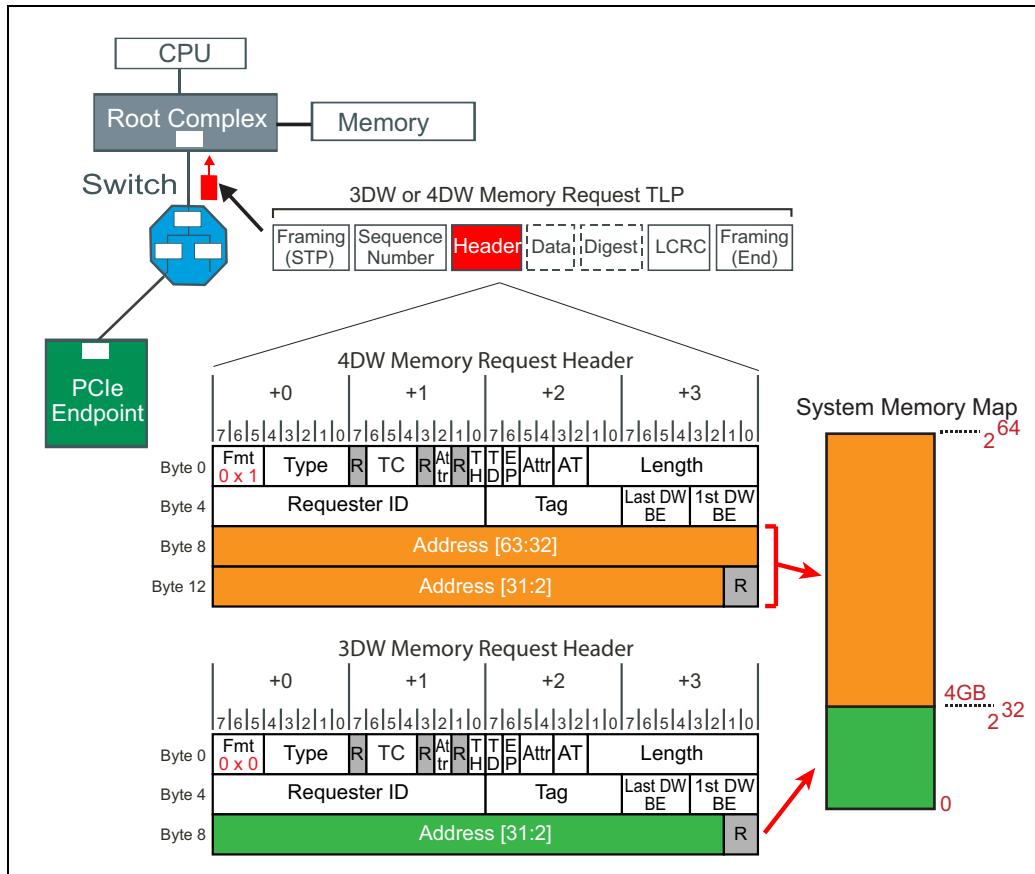
Field Name	Header Byte/Bit	Function
Requester ID [15:0]	Byte 4 Bit 7:0 Byte 5 Bit 7:0	Identifies the Requester's "return address" for corresponding Completion. Byte 4, 7:0 = Bus Number Byte 5, 7:3 = Device Number Byte 5, 2:0 = Function Number
Tag [7:0]	Byte 6 Bit 7:0	These bits identify the specific requests from the requester. A unique tag value is assigned to each outgoing Request. By default, only bits 4:0 are used, but the Extended Tag and Phantom Functions options can extend that to 11 bits, permitting up to 2048 outstanding requests to be in progress simultaneously.
Last DW BE [3:0] (Last DW Byte Enables)	Byte 7 Bit 7:4	These bits must be 0000b because IO requests can only be one DW in size.
1st DW BE [3:0] (First DW Byte Enables)	Byte 7 Bit 3:0	These bits qualify the bytes in the one-DW payload. For IO requests, any bit combination is valid (including all zeros).
Address [31:2]	Byte 8 Bit 7:0 Byte 9 Bit 7:0 Byte 10 Bit 7:0 Byte 11 Bit 7:2	The upper 30 bits of the 32-bit start address for the IO transfer. The lower two bits of the 32 bit address are reserved (00b), forcing a DW-aligned start address.

# PCI Express Technology

## Memory Requests

PCI Express memory transactions include two classes: Read Requests with their corresponding Completions, and Write Requests. The system memory map shown in Figure 5-8 on page 188 depicts both a 3DW and 4DW memory request packet. Keep in mind a point that the spec reiterates several times: a memory transfer is never permitted to cross a 4KB address boundary.

Figure 5-8: 3DW And 4DW Memory Request Header Formats



**Memory Request Header Fields.** The location and use of each field in a 4DW memory request header is listed in Table 5-5 on page 189. Note that the difference between a 3DW header and a 4DW header is simply the location and size of the starting Address field.

## Chapter 5: TLP Elements

---

Table 5-5: 4DW Memory Request Header Fields

Field Name	Header Byte/Bit	Function
Fmt [2:0] (Format)	Byte 0 Bit 7:5	Packet Formats: 000b = Memory Read (3DW w/o data) 010b = Memory Write (3DW w/ data) 001b = Memory Read (4DW w/o data) 011b = Memory Write (4DW w/ data) 1xxb = TLP Prefix has been added to the beginning of the packet. See “TPH (TLP Processing Hints)” on page 899 for more on this.
Type[4:0]	Byte 0 Bit 4:0	TLP packet Type field: 00000b = Memory Read or Write 00001b = Memory Read Locked Type field is used with Fmt [1:0] field to specify transaction type, header size, and whether data payload is present.
TC [2:0] (Traffic Class)	Byte 1 Bit 6:4	These bits encode the traffic class to be applied to a Request and to any associated Completion. 000b = Traffic Class 0 (Default)  .  111b = Traffic Class 7 See “Traffic Class (TC)” on page 247 for more on this.
Attr [2] (Attributes)	Byte 1 Bit 2	Indicates whether ID-based Ordering is to be used for this TLP. To learn more, see “ID Based Ordering (IDO)” on page 301.
TH (TLP Processing Hints)	Byte 1 Bit 0	Indicates whether TLP Hints have been included. See “TPH (TLP Processing Hints)” on page 899 for a discussion on these hints.

# PCI Express Technology

---

Table 5-5: 4DW Memory Request Header Fields (Continued)

Field Name	Header Byte/Bit	Function
TD (TLP Digest)	Byte 2 Bit 7	<p>If 1, the optional TLP Digest field is included with this TLP.</p> <p><u>Some rules:</u></p> <p>The presence of the Digest field must be checked by all receivers (using this bit)</p> <ul style="list-style-type: none"> <li>• TLPs with TD = 1 but no Digest field are treated as Malformed.</li> <li>• If the TD bit is set, recipient must perform the ECRC check if enabled.</li> <li>• If a Receiver doesn't support the optional ECRC checking, it must ignore the digest field.</li> </ul>
EP (Poisoned Data)	Byte 2 Bit 6	If 1, the data accompanying this packet should be considered to have an error although the transaction is allowed to complete normally.
Attr [1:0] (Attributes)	Byte 2 Bit 5:4	<p><u>Bit 5 = Relaxed ordering.</u></p> <p>When set = 1, PCI-X relaxed ordering is enabled for this TLP. Otherwise, strict PCI ordering is used.</p> <p><u>Bit 4 = No Snoop.</u></p> <p>If 1, system hardware is not required to cause processor cache snoop for coherency for this TLP. Otherwise, cache snooping is required.</p>
Address Type [1:0]	Byte 2 Bit 3:2	<p>This field supports address translation for virtualized systems. The translation protocol is described in a separate spec called <i>Address Translation Services</i>, where it can be seen that the field encodes as:</p> <ul style="list-style-type: none"> <li>00 = Default/Untranslated</li> <li>01 = Translation Request</li> <li>10 = Translated</li> <li>11 = Reserved</li> </ul>

## Chapter 5: TLP Elements

---

*Table 5-5: 4DW Memory Request Header Fields (Continued)*

Field Name	Header Byte/Bit	Function
Length [9:0]	Byte 2 Bit 1:0 Byte 3 Bit 7:0	TLP data payload transfer size, in DW. Maximum size is 1024 DW (4KB), encoded as:  00 0000 0001b = 1DW 00 0000 0010b = 2DW  .  11 1111 1111b = 1023 DW 00 0000 0000b = 1024 DW
Requester ID [15:0]	Byte 4 Bit 7:0 Byte 5 Bit 7:0	Identifies a Requester's return address for a completion: Byte 4, 7:0 = Bus Number Byte 5, 7:3 = Device Number Byte 5, 2:0 = Function Number
Tag [7:0]	Byte 6 Bit 7:0	These identify each outstanding request issued by the Requester. By default only bits 4:0 are used, allowing up to 32 requests to be in progress at a time. If the Extended Tag bit in the Control Register is set, then all 8 bits may be used (256 tags).
Last BE [3:0] (Last DW Byte Enables)	Byte 7 Bit 7:4	These qualify bytes within the last DW of data transferred.
1st DW BE [3:0] (First DW Byte Enables)	Byte 7 Bit 3:0	These qualify bytes within the first DW of the data payload.
Address [63:32]	Byte 8 Bit 7:0 Byte 9 Bit 7:0 Byte 10 Bit 7:0 Byte 11 Bit 7:0	The upper 32 bits of the 64-bit start address for the memory transfer.
Address [31:2]	Byte 12 Bit 7:0 Byte 13 Bit 7:0 Byte 14 Bit 7:0 Byte 15 Bit 7:2	The lower 32 bits of the 64 bit start address for the memory transfer. The lower two bits of the address are reserved, forcing a DW-aligned start address.

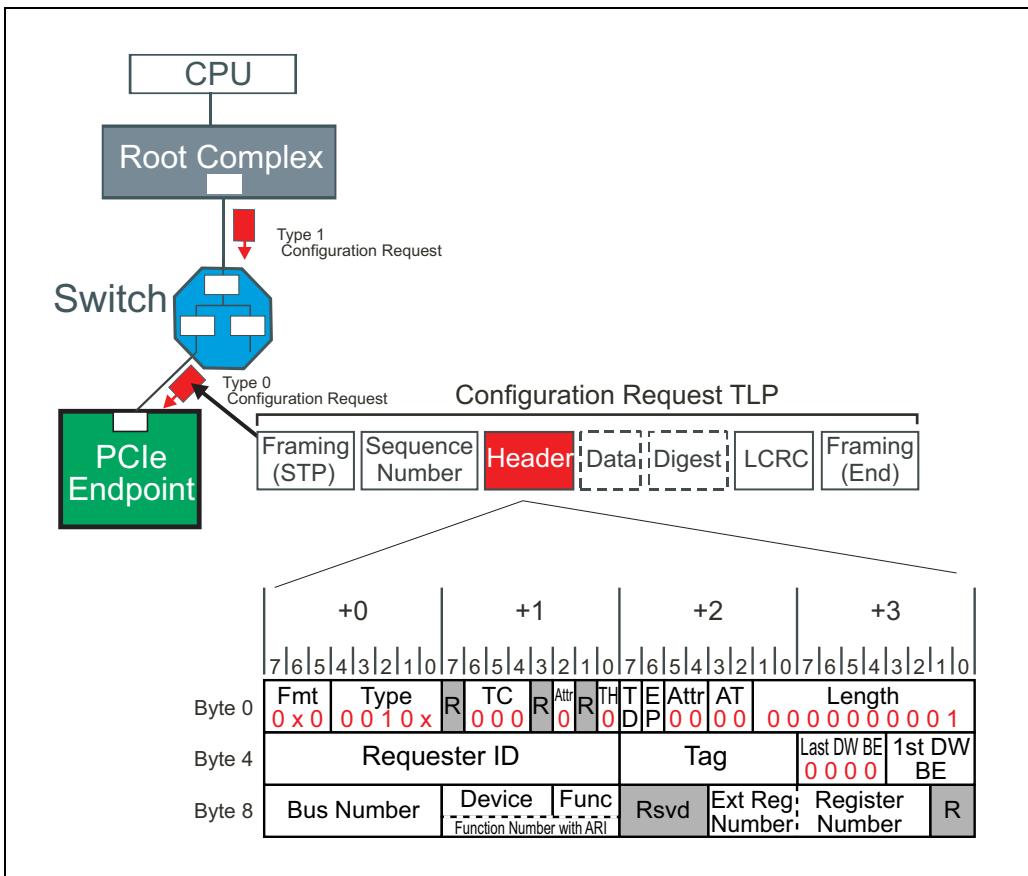
**Memory Request Notes.** Features of memory requests include:

1. Memory data transfers are not permitted to cross a 4KB boundary.
2. All memory-mapped writes are posted to improve performance.
3. Either 32- or 64-bit addressing may be used.
4. Data payload size is between 0 and 1024 DW (0-4KB).
5. Quality of Service features may be used, including up to 8 Traffic Classes.
6. The No Snoop attribute can be used to relieve the system of the need to snoop processor caches when transactions target main memory.
7. The Relaxed Ordering attribute may be used to allow devices in the packet's path to apply the relaxed ordering rules in hopes of improving performance.

## Configuration Requests

PCI Express uses both Type 0 and Type 1 configuration requests the same way PCI did to maintain backward compatibility. A Type 1 cycle propagates downstream until it reaches the bridge whose secondary bus matches the target bus. At that point, the configuration transaction is converted from Type 1 to Type 0 by the bridge. The bridge knows when to forward and convert configuration cycles based on the previously programmed bus number registers: Primary, Secondary, and Subordinate Bus Numbers. For more on this topic, refer to the section "Legacy PCI Mechanism" on page 91.

Figure 5-9: 3DW Configuration Request And Header Format



In Figure 5-9 on page 193, a Type 1 configuration cycle is shown making its way downstream, where it is converted to Type 0 by the bridge for that bus (accomplished by changing bit 0 of the Type field). Note that, unlike PCI, only one device can reside downstream on a Link. Consequently, no IDSEL or other hardware indication is needed to tell the device that it should claim the Type 0 cycle; any Type 0 configuration cycle a device sees on its Upstream Link will be understood as targeting that device.

**Definitions Of Configuration Request Header Fields.** Table 5-6 on page 194 describes the location and use of each field in the configuration request header illustrated in Figure 5-9 on page 193.

# PCI Express Technology

---

Table 5-6: Configuration Request Header Fields

Field Name	Header Byte/Bit	Function
Fmt [2:0] (Format)	Byte 0 Bit 7:5	Always a 3DW header 000b = configuration read (no data) 010b = configuration write (with data)
Type [4:0]	Byte 0 Bit 4:0	00100b = Type 0 Config Request 00101b = Type 1 Config Request
TC [2:0] (Transfer Class)	Byte 1 Bit 6:4	Traffic Class must be zero for Configuration requests, ensuring that these packets will never interfere with any high-priority packets.
Attr [2] (Attributes)	Byte 1 Bit 2	These bits are reserved and must be zero for Config Requests.
TH (TLP Processing Hints)	Byte 1 Bit 0	
TD (TLP Digest)	Byte 2 Bit 7	Indicates the presence of a digest field (1 DW) at the end of the TLP.
EP (Poisoned Data)	Byte 2 Bit 6	Indicates that data payload is poisoned.
Attr [1:0] (Attributes)	Byte 2 Bit 5:4	Relaxed Ordering and No Snoop bits are both always = 0 in configuration requests.
AT [1:0] (Address Type)	Byte 2 Bit 3:2	Address Type is reserved for config requests and must be zero.
Length [9:0]	Byte 2 Bit 1:0 Byte 3 Bit 7:0	Data payload size in DW is always = 1 for configuration requests. Byte Enables qualify bytes within the DW and any combination is legal.

## Chapter 5: TLP Elements

---

*Table 5-6: Configuration Request Header Fields (Continued)*

Field Name	Header Byte/Bit	Function
Requester ID [15:0]	Byte 4 Bit 7:0 Byte 5 Bit 7:0	Identifies the Requester's return address for a completion: Byte 4, 7:0 = Bus Number Byte 5, 7:3 = Device Number Byte 5, 2:0 = Function Number
Tag [7:0]	Byte 6 Bit 7:0	These bits identify outstanding request issued by the requester. By default, only bits 4:0 are used (32 outstanding transactions at a time), but if the Extended Tag bit in the Control Register is set = 1, then all 8 bits may be used (256 tags).
Last BE [3:0] (Last DW Byte Enables)	Byte 7 Bit 7:4	These qualify bytes in the last data DW transferred. Since config requests can only be one DW in size, these bits must be zero.
1st DW BE [3:0] (First DW Byte Enables)	Byte 7 Bit 3:0	These high-true bits qualify bytes in the first data DW transferred. For config requests, any bit combination is valid (including none active).
Completer ID [15:0]	Byte 8 Bit 7:0 Byte 9 Bit 7:0	Identifies the completer being accessed with this configuration cycle. Byte 8, 7:0 = Bus Number Byte 9, 7:3 = Device Number Byte 9, 2:0 = Function Number
Ext Register Number [3:0] (Extended Register Number)	Byte 10 Bit 3:0	These provide the upper 4 bits of DW space for accessing the extended config space. They're combined with Register Number to create the 10-bit address needed to access the 1024 DW (4096 byte) space. For PCI-compatible config space, this field must be zero.

# PCI Express Technology

---

Table 5-6: Configuration Request Header Fields (Continued)

Field Name	Header Byte/Bit	Function
Register Number [5:0]	Byte 11 Bit 7:0	As the lower 8 bits of configuration DW space, these specify the register number. The two lowest bits are always zero, forcing DW-aligned accesses.

**Configuration Request Notes.** Configuration requests always use the 3DW header format and are routed based on the target Bus, Device and Function numbers. All devices “capture” their Bus and Device Number from the target numbers in the Request whenever they receive a Type 0 configuration write cycle. The reason for that is because they’ll need it later to use as their Requester ID when they send requests of their own in the future.

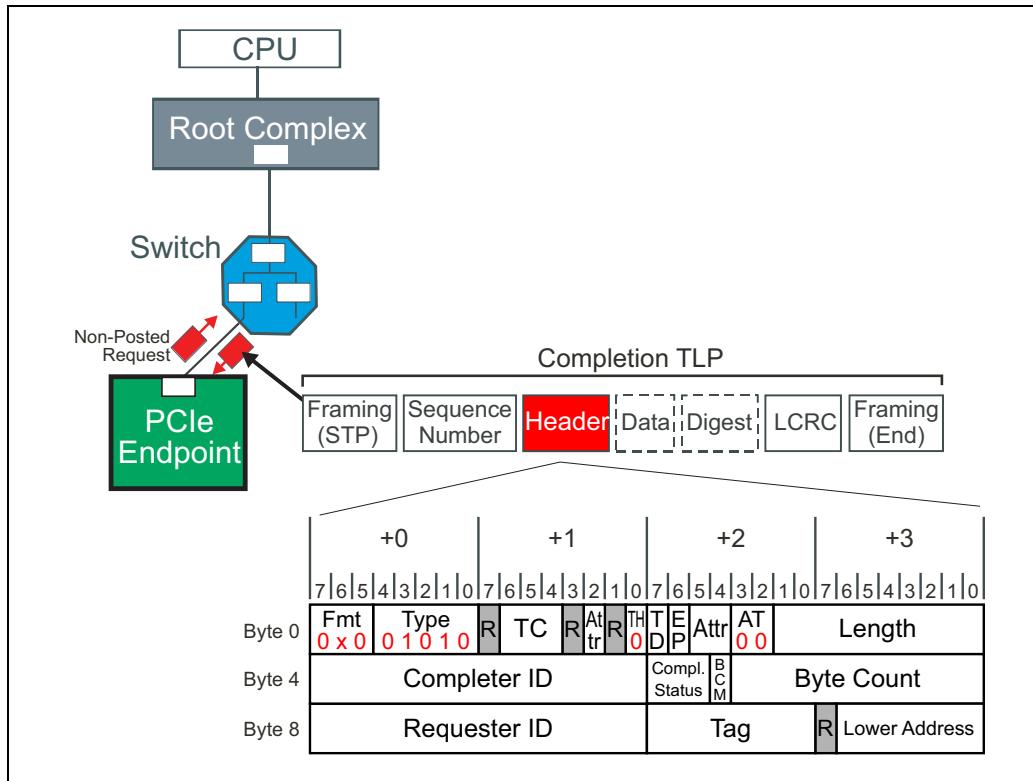
## Completions

Completions are expected in response to non-posted Request, unless errors prevent them. For example Memory, IO, or Configuration Read requests usually result in Completions with data. On the other hand, IO or Configuration Write requests usually result in a completion without data that merely reports the status of the transaction.

Many fields in the Completion use the same values as the associated request, including Traffic Class, Attribute bits, and the original Requester ID (used to route the completion back to the Requester). Figure 5-10 on page 197 shows a completion returned for a non-posted request, and the 3DW header format it uses. Completions also supply the Completer ID in the header. Completer ID is not interesting during normal operation, but knowing where the Completion came from could be useful for error diagnosis during system debug.

## Chapter 5: TLP Elements

Figure 5-10: 3DW Completion Header Format



**Definitions Of Completion Header Fields.** Table 5-7 on page 197 describes the location and use of each field in a completion header.

Table 5-7: Completion Header Fields

Field Name	Header Byte/Bit	Function
Fmt [2:0] (Format)	Byte 0 Bit 7:5	Packet Format (always a 3DW header) 000b = Completion without data (Cpl) 010b = Completion with data (CpID)
Type [4:0]	Byte 0 Bit 4:0	Packet type is 01010b for Completions.

# PCI Express Technology

---

Table 5-7: Completion Header Fields (Continued)

Field Name	Header Byte/Bit	Function
TC [2:0] (Traffic Class)	Byte 1 Bit 6:4	Completions must use the same value here as the corresponding Request.
Attr [2] (Attributes)	Byte 1 Bit 2	Indicates whether ID-based Ordering is to be used for this TLP. To learn more, see “ID Based Ordering (IDO)” on page 301.
TH (TLP Processing Hints)	Byte 1 Bit 0	Reserved for Completions.
TD (TLP Digest)	Byte 2 Bit 7	If = 1, indicates the presence of a digest field at the end of the TLP.
EP (Poisoned Data)	Byte 2 Bit 6	If = 1, indicates the data payload is poisoned.
Attr [1:0] (Attributes)	Byte 2 Bit 5:4	Completions must use the same values here as the corresponding Request.
AT [1:0] (Address Type)	Byte 2 Bit 3:2	Address Type is reserved for Completions and must be zero, but Receivers are not required or even encouraged to check this.
Length [9:0]	Byte 2 Bit 1:0 Byte 3 Bit 7:0	Indicates data payload size in DW. For Completions, this field reflects the size of the data payload associated with this completion.
Completer ID [15:0]	Byte 4 Bit 7:0 Byte 5 Bit 7:0	Identifies the Completer to support debugging problems. Byte 4 7:0 = Completer Bus # Byte 5 7:3 = Completer Dev # Byte 5 2:0 = Completer Function #

## Chapter 5: TLP Elements

---

*Table 5-7: Completion Header Fields (Continued)*

Field Name	Header Byte/Bit	Function
Compl. Status [2:0] (Completion Status Code)	Byte 6 Bit 7:5	These bits indicate status for this Completion. 000b = Successful Completion (SC) 001b = Unsupported Request (UR) 010b = Config Req Retry Status (CRS) 100b = Completer abort (CA) All other codes are reserved. See “Summary of Completion Status Codes” on page 200.
BCM (Byte Count Modified)	Byte 6 Bit 4	This is only used by PCI-X Completers and indicates that the Byte Count field reports only the first payload rather than the total payload remaining. See “Using The Byte Count Modified Bit” on page 201.
Byte Count [11:0]	Byte 6 Bit 3:0 Byte 7 Bit 7:0	Byte count remaining to satisfy a read request, as derived from the original request Length field. See “Data Returned For Read Requests:” on page 201 for special cases caused by multiple completions.
Requester ID [15:0]	Byte 8 Bit 7:0 Byte 9 Bit 7:0	Copied from the Request for use as the return address (target) for this Completion. Byte 8, 7:0 = Requester Bus # Byte 9, 7:3 = Requester Device # Byte 9, 2:0 = Requester Function #
Tag [7:0]	Byte 10 Bit 7:0	This must be the Tag value received with the Request. Requester associates this Completion with a pending Request based on the Tag.

# PCI Express Technology

---

Table 5-7: Completion Header Fields (Continued)

Field Name	Header Byte/Bit	Function
Lower Address [6:0]	Byte 11 Bit 6:0	The lower 7 bits of address for the first data returned for a read request. Calculated from Request Length and Byte Enables, it assists buffer management by showing how many bytes can be transferred before reaching the next Read Completion Boundary. See “Calculating Lower Address Field” on page 200.

## Summary of Completion Status Codes.

- 000b (SC) Successful Completion: the Request was serviced properly.
- 001b (UR) Unsupported Request: Request is not legal or was not recognized by the Completer. This is an error condition but how the Completer responds depends on the spec revision to which it was designed. Before the 1.1 spec, this were considered an uncorrectable error, but for 1.1 and later it's treated as an Advisory Non-Fatal Error. See the “Unsupported Request (UR) Status” on page 663 for details.
- 010b (CRS) Configuration Request Retry Status: Completer is temporarily unable to service a configuration request, and the request should be attempted again later.
- 100b (CA) Completer Abort: Completer should have been able to service the request but has failed for some reason. This is an uncorrectable error.

**Calculating The Lower Address Field .** This field is set up by the Completer to reflect the byte-aligned address of the first enabled byte of data being returned in the Completion payload. Hardware calculates this by considering both the DW start address and the Byte Enable pattern in the First DW Byte Enable field provided in the original request.

For Memory Read Requests, the address is an offset from the DW start address:

- If the First DW Byte Enable field is 1111b, all bytes are enabled in the first DW and the offset is 0. This field matches the DW-aligned start address.
- If the First DW Byte Enable field is 1110b, the upper three bytes are enabled in the first DW and the offset is 1. This field is the DW start address + 1.
- If the First DW Byte Enable field is 1100b, the upper two bytes are enabled

- in the first DW and the offset is 2. This field is the DW start address + 2.
- If the First DW Byte Enable field is 1000b, only the upper byte is enabled in the first DW and the offset is 3. This field is the DW start address + 3.

Once calculated, the lower 7 bits are placed in the Lower Address field of the Completion header to facilitate the case in which the read completion is smaller than the entire payload and needs to stop at the first RCB. Breaking a transaction must be done on RCBs, and the number of bytes transferred to reach the first one is based on start address.

For AtomicOp Completions, the Lower Address field is reserved. For all other Completion types, it's set to zero.

**Using The Byte Count Modified Bit.** This bit is only set by PCI-X Completers, but they could exist in a PCIe topology if a bridge from PCIe to PCI-X is used. Rules for its assertion include:

- It's only set by a PCI-X Completer if a read request is going to be broken into multiple completions.
- It's only set for the first Completion of the series, and only then to indicate that the first Completion contains a Byte Count field that reflects the first Completion payload rather than the total remaining (as it normally would). The Requester understands that, even though the Byte Count appears to show that this is the last Completion for this request, this Completion will instead be followed by others to satisfy the original request as required.
- For subsequent Completions in the series, the BCM bit must be deasserted and the Byte Count field will reflect the total remaining count as it normally would.
- Devices receiving Completions with the BCM bit set must interpret this case properly.
- The Lower Address field is set by the Completer during completions with data to reflect the address of the first enabled byte of data being returned

## Data Returned For Read Requests:

- A read request may require multiple completions to be fulfilled, but total data transfer must eventually equal the size of original request, or a Completion Timeout error will probably result.
- A given Completion can only service one Request.
- IO and Configuration reads are always 1 DW, and will always be satisfied with a single Completion
- A Completion with a Status Code other than SC (successful) terminates a transaction.

5. The Read Completion Boundary (RCB) must be observed when handling a read request with multiple completions. The RCB is 64 bytes or 128 bytes for the Root Complex, since it is allowed to modify the size of packets flowing between its ports, and the value used is visible in a configuration register.
6. Bridges and endpoints may implement a bit for selecting the RCB size (64 or 128 bytes) under software control.
7. Completions that are entirely within an aligned RCB boundary must complete in one transfer, since the transfer won't reach the RCB, which is the only place it can legally stop early.
8. Multiple Completions for a single read request must return data in increasing address order.

## Receiver Completion Handling Rules:

1. A received Completion that doesn't match a pending request is an Unexpected Completion and treated as an error.
2. Completions with a completion status other than SC or CRS will be handled as errors and buffer space associated with them will be released.
3. When the Root Complex receives a CRS status during a configuration cycle, the request is terminated. What happens next is implementation specific, but if the Root supports it, the action is defined by the setting of its CRS Software Visibility bit in the Root Control register.
  - If CRS Software Visibility is not enabled, the Root will reissue the config request for an implementation-specific number of times before giving up and concluding the target has a problem.
  - If CRS Software Visibility is enabled, software designed to support it will always read both bytes of the Vendor ID field first. If the hardware then receives a CRS for that Request, it returns the value 0001h for the Vendor ID. This value, reserved for this use by the PCI-SIG, doesn't correspond to any valid Vendor ID and informs software about this event. This allows software to go on to some other task while waiting for the target to become ready (which could take as long as 1 second after reset) rather than being stalled. Any other config read or write will simply be automatically retried by the Root as a new Request for the design-specific number of iterations.
4. A CRS status in response to a request other than configuration is illegal and may be reported as a Malformed TLP.
5. Completions with status = reserved code are treated as if the code was UR.
6. If a Read Completion or an AtomicOp Completion is received with a status other than SC, no data is included with the completion and the Requester must consider this Request terminated. How the Requester handles this case is implementation-specific.

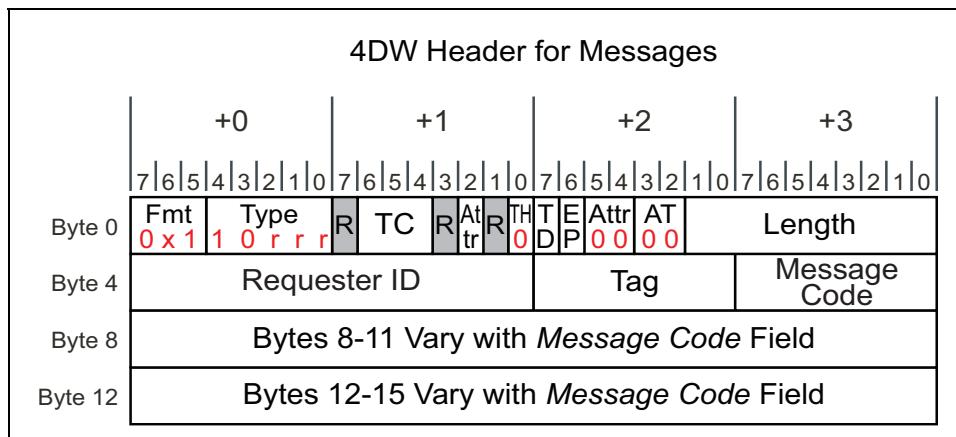
## Chapter 5: TLP Elements

7. In the event multiple completions are being returned for a read request, a completion status other than SC ends the transaction. Device handling of data received prior to the error is implementation-specific.
  8. For compatibility with PCI, a Root Complex may be required to synthesize a read value of all "1's" when a configuration cycle ends with a completion indicating an Unsupported Request. This is analogous to a PCI Master Abort that happens when enumeration software attempts to read from devices that are not present.

## Message Requests

Message Requests replace many of the interrupt, error, and power management sideband signals used on PCI and PCI-X. All Message Requests use the 4DW header format, but not all of the fields are used in every Message type. Fields in bytes 8 through 15 are not defined for some Messages and are reserved for those cases. Messages are treated much like posted Memory Write transactions but their routing can be based on address, ID, and in some cases the routing can be implicit. The *routing subfield* (Byte 0, bits 2:0) in the packet header indicates which routing method is used and which additional header registers are defined. The general Message Request header format is shown in Figure 5-11 on page 203.

Figure 5-11: 4DW Message Request Header Format



# PCI Express Technology

---

## Message Request Header Fields.

Table 5-8: Message Request Header Fields

Field Name	Header Byte/ Bit	Function
Fmt [2:0] (Format)	Byte 0 Bit 7:5	Packet Format. Always a 4DW header 001b = Message Request without data 011b = Message Request with data
Type [4:0]	Byte 0 Bit 4:0	TLP packet type field. Set to: Bit 4:3: 10b = Msg <u>Bit 2:0</u> (Message Routing Subfield) 000b = Implicitly Routed to RC (Root Complex) 001b = Routed by address 010b = Routed by ID 011b = Implicitly Broadcast from RC 100b = Local; terminate at receiver 101b = Gather & route to RC Others = Reserved, treated as Local
TC [2:0] (Traffic Class)	Byte 1 Bit 6:4	TC is always zero for most Message Requests, ensuring that they don't interfere with high-priority packets.
Attr [2] (Attributes)	Byte 1 Bit 2	Indicates whether ID-based Ordering is to be used for this TLP. To learn more, see "ID Based Ordering (IDO)" on page 301.
TH (TLP Processing Hints)	Byte 1 Bit 0	Reserved, except as noted.
TD	Byte 2 Bit 7	If = 1, indicates the presence of a digest field (1 DW) at the end of the TLP (preceding LCRC and END)
EP	Byte 2 Bit 6	If = 1, indicates the data payload (if present) is poisoned.

## Chapter 5: TLP Elements

---

*Table 5-8: Message Request Header Fields (Continued)*

Field Name	Header Byte/ Bit	Function
Attr [1:0] (Attributes)	Byte 2 Bit 5:4	Except as noted, these are always reserved in Message Requests.
AT [1:0] (Address Type)	Byte 2 Bit 3:2	Address Type is reserved for Messages and must be zero, but Receivers are not required or even encouraged to check this.
Length [9:0]	Byte 2 Bit 1:0 Byte 3 Bit 7:0	Indicates data payload size in DW. For Message Requests, this field is always 0 (no data) or 1 (one DW of data)
Requester ID [15:0]	Byte 4 Bit 7:0 Byte 5 Bit 7:0	Identifies the Requester sending the message. Byte 4, 7:0 = Requester Bus # Byte 5, 7:3 = Requester Device # Byte 5, 2:0 = Requester Function #
Tag [7:0]	Byte 6 Bit 7:0	Since all Message Requests are posted and don't receive Completions, no tag is assigned to them. These bits should be zero.
Message Code [7:0]	Byte 7 Bit 7:0	This field contains the code indicating the type of message being sent. 0000 0000b = Unlock Message 0001 0000b = Lat. Tolerance Reporting 0001 0010b = Optimized Buffer Flush/Fill 0001 xxxx b = Power Mgt. Message 0010 0xxx b = INTx Message 0011 00xx b = Error Message 0100 xxxx b = Ignored Messages 0101 0000b = Set Slot Power Message 0111 111x b = Vendor-Defined Messages

# PCI Express Technology

---

Table 5-8: Message Request Header Fields (Continued)

Field Name	Header Byte/ Bit	Function
Address [63:32]	Byte 8 Bit 7:0 Byte 9 Bit 7:0 Byte 10 Bit 7:0 Byte 11 Bit 7:0	If address routing was selected for the message (see Type 4:0 field above), then this field contains the upper 32 bits of the 64 bit starting address. Otherwise, this field is not used.
Address [31:2]	Byte 12 Bit 7:0 Byte 13 Bit 7:0 Byte 14 Bit 7:0 Byte 15 Bit 7:2	If address routing is selected (see Type field above), then this field contains the lower part of the 64-bit starting address. If ID routing is selected, Bytes 8 and 9 form the target ID. Otherwise, this field is not used.

**Message Notes:** The following tables specify the message coding used for each of the nine message groups, and is based on the message code field listed in Table 5-8 on page 204. The defined message groups include:

1. INTx Interrupt Signaling
2. Power Management
3. Error Signaling
4. Locked Transaction Support
5. Slot Power Limit Support
6. Vendor-Defined Messages
7. Ignored Messages (related to Hot-Plug support in spec revision 1.1)
8. Latency Tolerance Reporting (LTR)
9. Optimized Buffer Flush and Fill (OBFF)

**INTx Interrupt Messages.** Many devices are capable of using the PCI 2.3 Message Signaled Interrupt (MSI) method of delivering interrupts, but older devices may not support it. For these cases, PCIe defines a “virtual wire” alternative in which devices simulate the assertion and deassertion of the PCI interrupt pins (INTA-INTD) by sending Messages. The interrupting device sends the first Message to inform the upstream device that an interrupt has been asserted. Once the interrupt has been serviced, the interrupting device sends a second Message to communicate that the signal has been released. For more on this protocol, refer to the section called “Virtual INTx Signaling” on page 805 for details.

## Chapter 5: TLP Elements

---

Table 5-9: INTx Interrupt Signaling Message Coding

INTx Message	Message Code 7:0	Routing 2:0
Assert_INTA	0010 0000b	100b (Local - Terminate at Rx)
Assert_INTB	0010 0001b	
Assert_INTC	0010 0010b	
Assert_INTD	0010 0011b	
Deassert_INTA	0010 0100b	
Deassert_INTB	0010 0101b	
Deassert_INTC	0010 0110b	
Deassert_INTD	0010 0111b	

Rules regarding the use of INTx Messages:

1. They have no data payload and so the Length field is reserved.
2. They're only issued by Upstream Ports. Checking this rule for received packets is optional but, if checked, violations will be handled as Malformed TLPs.
3. They're required to use the default traffic class TC0. Receivers must check for this and violations will be handled as Malformed TLPs.
4. Components at both ends of the Link must track the current state of the four virtual interrupts. If the logical state of one interrupt changes at the Upstream Port, it must send the appropriate INTx message.
5. INTx signaling is disabled when the Interrupt Disable bit of the Command Register is set = 1 (as would be the case for physical interrupt lines).
6. If any virtual INTx signals are active when the Interrupt Disable bit is set in the device, the Upstream Port must send corresponding Deassert\_INTx messages.
7. Switches must track the state of the four INTx signals independently for each Downstream Port and combine the states for the Upstream Port.
8. The Root Complex must track the state of the four INTx lines independently and convert them into system interrupts in an implementation-specific way.

# PCI Express Technology

---

9. They use the routing type “Local-Terminate at Receiver” to allow a Switch to remap the designated interrupt pin when necessary (see “Mapping and Collapsing INTx Messages” on page 808). Consequently, the Requester ID in an INTx message may be assigned by the last transmitter.

**Power Management Messages.** PCI Express is compatible with PCI power management, and adds hardware-based Link power management as well. Messages are used to convey some of this information, but to learn how the overall PCIe power management protocol works, refer to Chapter 16, entitled "Power Management," on page 703. Table 5-10 on page 208 summarizes the four power management message types.

*Table 5-10: Power Management Message Coding*

Power Management Message	Message Code 7:0	Routing 2:0
PM_Active_State_Nak	0001 0100b	100b
PM_PME	0001 1000b	000b
PM_Turn_Off	0001 1001b	011b
PME_TO_Ack	0001 1011b	101b

Power Management Message Rules:

1. Power Management Messages don't have a data payload, so the Length field is reserved.
2. They're required to use the default traffic class TC0. Receivers must check for this and handle violations as Malformed TLPs.
3. PM\_Active\_State\_Nak is sent from a Downstream Port after it observes a request from the Link neighbor to change the Link power state to L1 but it has chosen not to do so (Local - Terminate at Receiver routing).
4. PM\_PME is sent upstream by the component requesting a Power Management Event (Implicitly Routed to the Root Complex).
5. PM\_Turn\_Off is sent downstream to all endpoints (Implicitly Broadcast from the Root Complex routing).
6. PME\_TO\_Ack is sent upstream by endpoints. For switches with multiple Downstream Ports, this message won't be forwarded upstream until all Downstream Ports have received it (Gather and Route to the Root Complex routing).

## Chapter 5: TLP Elements

---

**Error Messages.** Error Messages are sent upstream (Implicitly Routed to the Root Complex) by enabled components that detect errors. To assist software in knowing how to service the error, the Error Message identifies the requesting agent in the Requester ID field of the message header. Table 5-11 on page 209 describes the three error message types.

*Table 5-11: Error Message Coding*

Error Message	Message Code 7:0	Routing 2:0
ERR_COR (Correctable)	0011 0000b	000b
ERR_NONFATAL (Uncorrectable, Non-fatal)	0011 0001b	
ERR_FATAL (Uncorrectable, Fatal)	0011 0011b	

Error Signaling Message Rules:

1. They're required to use the default traffic class TC0. Receivers must check for this and handle violations as Malformed TLPs.
2. They don't have a data payload, so the Length field is reserved.
3. The Root Complex converts Error Messages into system-specific events.

**Locked Transaction Support.** The Unlock Message is used as part of the Locked transaction protocol defined for PCI and still available to Legacy Devices. The protocol begins with a Memory Read Locked Request. When that Request is seen by Ports along the path to the target device, they implement an atomic read-modify-write protocol by locking out other Requesters from using VC0 until the Unlock Message is received. This Message is sent to the target to release all the Ports in the path to it and finish the Locked Transaction sequence. Table 5-12 on page 209 summarizes the coding for this message.

*Table 5-12: Unlock Message Coding*

Unlock Message	Message Code 7:0	Routing 2:0
Unlock	0000 0000b	011b

# PCI Express Technology

---

Unlock Message Rules:

1. They're required to use the default traffic class TC0. Receivers must check for this and handle violations as Malformed TLPs.
2. They don't have a data payload, and the Length field is reserved.

**Set Slot Power Limit Message.** This is sent from a Downstream Port to the device plugged into the slot. This power limit is stored in the endpoint in its Device Capabilities Register. Table 5-13 summarizes the message coding.

*Table 5-13: Slot Power Limit Message Coding*

Slot Power Limit Message	Message Code 7:0	Routing 2:0
Set_Slot_Power_Limit	0101 0000b	100b

Set\_Slot\_Power\_Limit Message Rules:

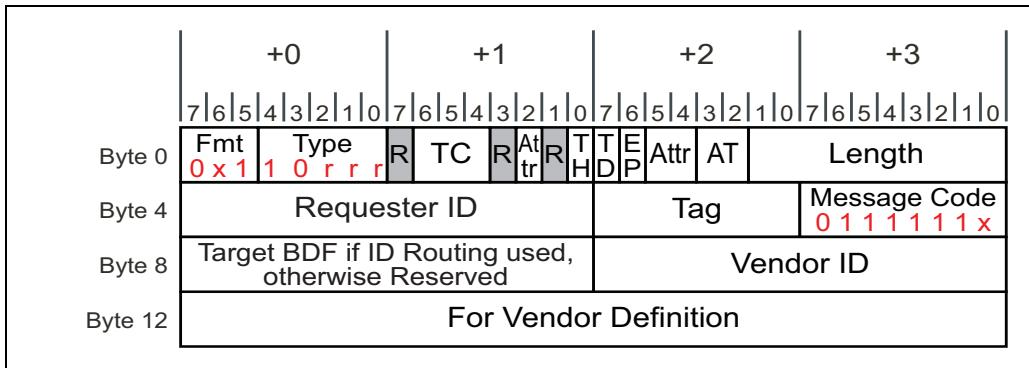
1. They're required to use the default traffic class TC0. Receivers must check for this and handle violations as Malformed TLPs.
2. The data payload is 1 DW and so the Length field is set to one. Only the lower 10 bits of the 32-bit data payload are used for slot power scaling; the upper payload bits must be set to zero.
3. This message is sent automatically anytime the Data Link Layer transitions to DL\_Up status or if a configuration write to the Slot Capabilities Register occurs while the Data Link Layer is already reporting DL\_Up status.
4. If the card in the slot already consumes less power than the power limit specified, it's allowed to ignore the Message.

**Vendor-Defined Message 0 and 1.** These are intended to allow expansion of the PCIe messaging capabilities either by the spec or by vendor-specific extensions. The header for them is shown in Figure 5-12 on page 211, and the codes are given in Figure 5-14 on page 211.

## Chapter 5: TLP Elements

---

*Figure 5-12: Vendor-Defined Message Header*



*Table 5-14: Vendor-Defined Message Coding*

Vendor-Defined Message	Message Code 7:0	Routing 2:0
Vendor Defined Message 0	0111 1110b	000b, 010b, 011b, 100b
Vendor Defined Message 1	0111 1111b	

Vendor-Defined Message Rules:

1. A data payload may or may not be included with either type.
2. Messages are distinguished by the Vendor ID field.
3. Attribute bits [2] and [1:0] are not reserved.
4. If the Receiver doesn't recognize the Message:
  - Type 1 Messages are silently discarded
  - Type 0 Messages are treated as an Unsupported Request error condition

**Ignored Messages.** Listing an entire category of Messages that are to be ignored sounds a little strange without the context for it. These were formerly Hot Plug Signaling messages that supported devices that had Hot Plug indicators and push buttons on the add-in card itself rather than on the system board. This Message type was defined through spec rev 1.0a, but this option was no longer supported beginning with the 1.1 spec release, so the details are only included here for reference. As the name now suggests, Transmitters are

# PCI Express Technology

---

strongly encouraged not to send these messages, and Receivers are strongly encouraged to ignore them if they are seen. If they're still going to be used anyway, they must conform to the 1.0a spec details.

Table 5-15: Hot Plug Message Coding

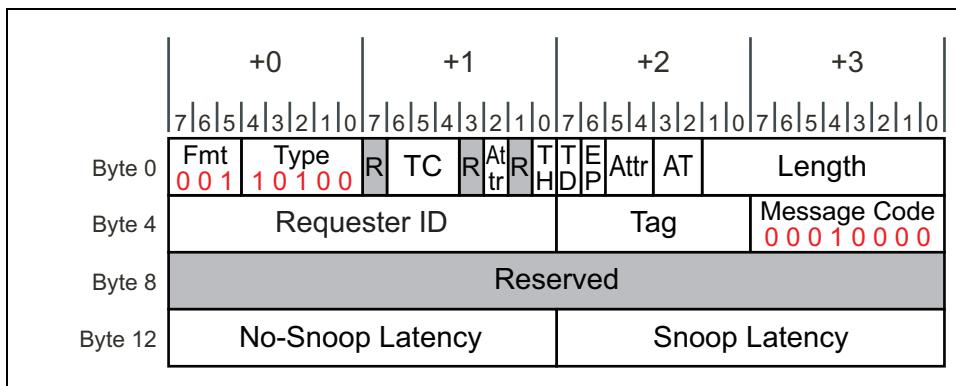
Error Message	Message Code 7:0	Routing 2:0
Attention_Indicator_On	0100 0001b	100b
Attention_Indicator_Blink	0100 0011b	100b
Attention_Indicator_Off	0100 0000b	100b
Power_Indicator_On	0100 0101b	100b
Power_Indicator_Blink	0100 0111b	100b
Power_Indicator_Off	0100 0100b	100b
Attention_Button_Pressed	0100 1000b	100b

Hot Plug Message Rules:

- They are driven by a Downstream Port to the card in the slot.
- The Attention Button Message is driven upstream by a slot device.

**Latency Tolerance Reporting Message.** LTR Messages are used to optionally report acceptable read/write service latencies for a device. To learn more about this power management technique, see the section called “LTR (Latency Tolerance Reporting)” on page 784.

Figure 5-13: LTR Message Header



## Chapter 5: TLP Elements

Table 5-16: LTR Message Coding

Latency Tolerance Reporting Message	Message Code 7:0	Routing 2:0
LTR	0001 0000	100

LTR Message Rules:

1. They're required to use the default traffic class TC0. Receivers must check for this and handle violations as Malformed TLPs.
2. They don't have a data payload, and the Length field is reserved.

**Optimized Buffer Flush and Fill Messages.** OBFF Messages are used to report platform power status to Endpoints and facilitate more effective system power management. To learn more about this technique, see the discussion called "OBFF (Optimized Buffer Flush and Fill)" on page 776.

Figure 5-14: OBFF Message Header

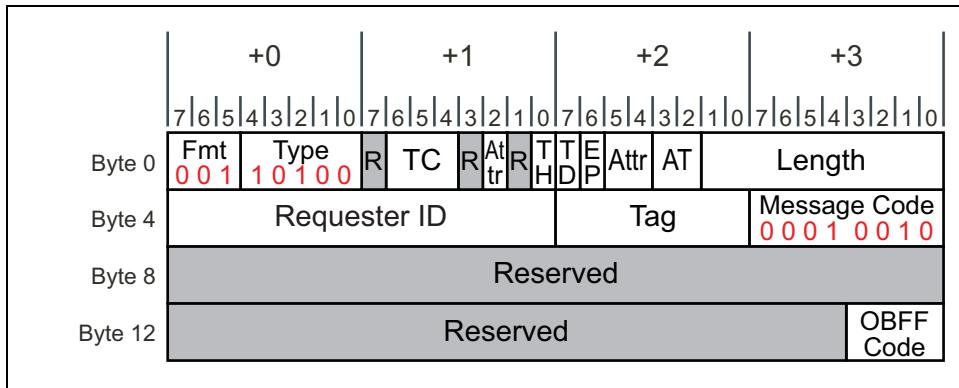


Table 5-17: LTR Message Coding

Optimized Buffer Flush/Fill Message	Message Code 7:0	Routing 2:0
OBFF	0001 0010	100

# **PCI Express Technology**

---

OBFF Message Rules:

1. They're required to use the default traffic class TC0. Receivers must check for this and handle violations as Malformed TLPs.
2. They don't have a data payload, and the Length field is reserved.
3. The Requester ID must be set to the Transmitting Port's ID.

---

# 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.

# PCI Express Technology

---

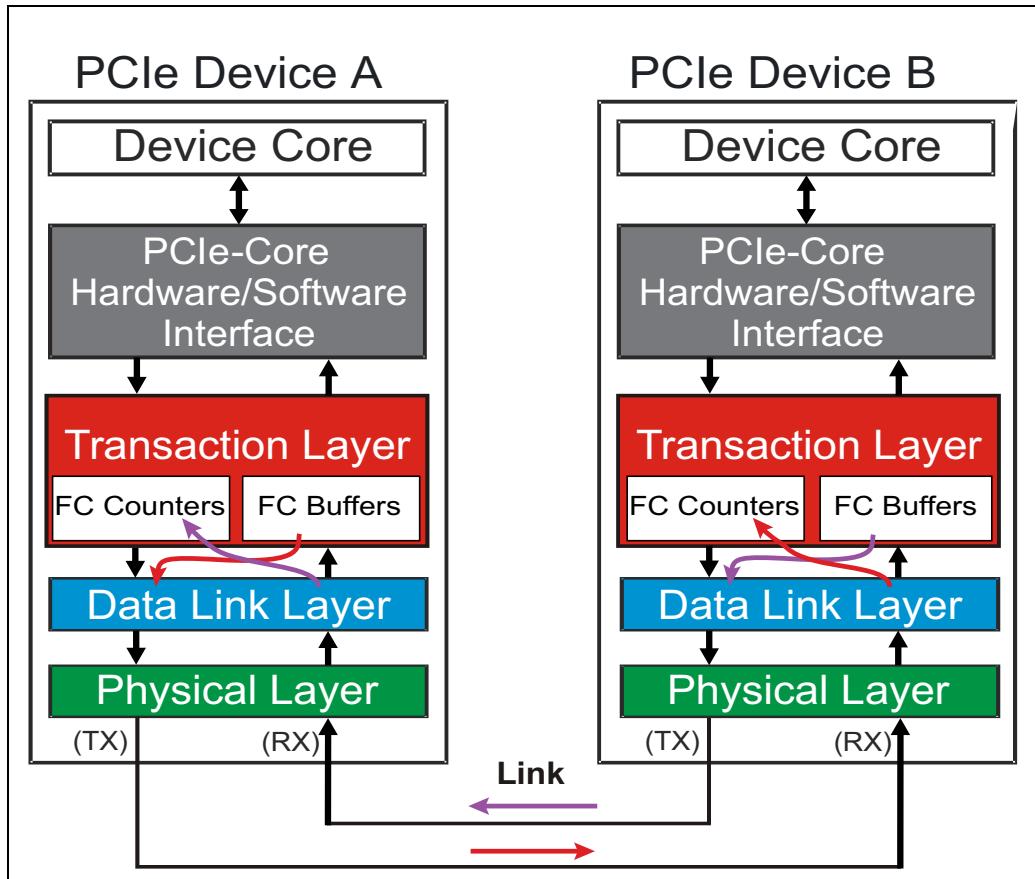
Flow Control mechanisms can improve transmission efficiency if multiple Virtual Channels (VCs) are used. Each Virtual Channel carries transactions that are independent from the traffic flowing in other VCs because flow-control buffers are maintained separately. Therefore, a full Flow Control buffer in 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. This 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.

# PCI Express Technology

## VC Flow Control Buffer Organization

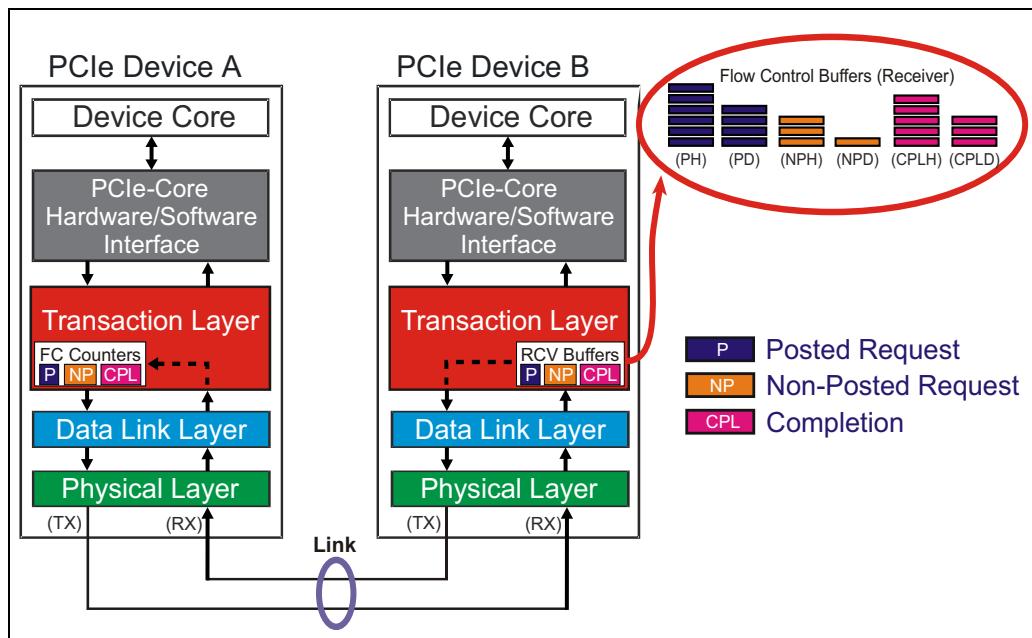
Each VC Flow Control buffer at the receiver is managed for each category of transaction flowing through the virtual channel. These categories are:

- 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



## Flow Control Credits

Buffer space is reported by the receiver in units called Flow Control credits. The unit value of Flow Control Credits (FCCs) for header and data buffers are:

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

Flow Control DLLPs communicate this information, and do not require Flow Control credits themselves. That's because they originate and terminate at the Link Layer and don't use the Transaction Layer buffers.

---

## Initial Flow Control Advertisement

During Flow Control initialization, PCIe devices communicate their buffer sizes by “advertising” their buffer space via flow control credits. PCIe also defines an infinite Flow Control credit value that is required for some buffers. A receiver that advertises infinite buffer space is effectively guaranteeing that its buffer space will never overflow.

---

## Minimum and Maximum Flow Control Advertisement

The specification defines the minimum number of credits that can be reported for the different Flow Control buffer types as listed in Table 6-1. However, devices normally advertise considerably more credits than the minimum. Table 6-2 on page 220 lists the maximum advertisement allowed by the specification.

*Table 6-1: Required Minimum Flow Control Advertisements*

Credit Type	Minimum Advertisement
Posted Request Header (PH)	1 unit. Credit Value = one 4DW HDR + Digest = 5DW.
Posted Request Data (PD)	Largest possible setting of the Max_Payload_Size in credits. Example: If the largest Max_Payload_Size value supported is 1024 bytes, the smallest permitted initial credit value would be 040h.

# PCI Express Technology

---

*Table 6-1: Required Minimum Flow Control Advertisements (Continued)*

Credit Type	Minimum Advertisement
Non-Posted Request HDR (NPH)	<b>1 unit.</b> Credit Value = one 4 DW HDR + Digest = 5DW.
Non-Posted Request Data (NPD)	<b>1 unit.</b> Credit Value = 4DW.  <b>2 unit.</b> Receivers supporting AtomicOp routing or AtomicOp Completer capability have credit value of 02h
Completion HDR (CPLH)	<b>1 unit.</b> Credit Value = one 3DW HDR + Digest = 4DW; for Root Complex with peer-to-peer support and Switches.  <b>Infinite units.</b> Initial Credit Value = all 0's for Root Complex with no peer-to-peer support and Endpoints.
Completion Data (CPLD)	<b>n unit.</b> Value of largest possible setting of Max_Payload_Size or size of largest Read Request (which ever is smaller) divided by FC Unit Size (4DW); for Root Complex with peer-to-peer support and Switches.  <b>Infinite units.</b> Initial Credit Value = all 0's; for Root Complex with no peer-to-peer support and Endpoints.

*Table 6-2: Maximum Flow Control Advertisements*

Credit Type	Maximum Advertisement
Posted Request Header (PH)	<b>128 units.</b> 128 credits @ 5 DWs = 2,560 bytes.
Posted Request Data (PD)	2048 units. Value of the Max_Payload_Size (4096 bytes) including all functions supported by device (8) divided by the credit size (4 DWs) = 32,768 bytes 2048 credits @ 4 DWs = 32,768 bytes
Non-Posted Request HDR (NPH)	<b>128 units.</b> 128 credits @ 5 DWs = 2,560 bytes.
Non-Posted Request Data (NPD)	The author's could not find a precise value for the maximum number of credits for Non-Posted Data. The maximum number of credits listed for Data is 2048. However, a more reasonable approach might use the Non-Posted header limit of 128 credits, because Non-Posted Data is always associated with Non-Posted Headers.

Table 6-2: Maximum Flow Control Advertisements (Continued)

Credit Type	Maximum Advertisement
Completion HDR (CPLH)	<b>128 units.</b> 128 credits @ 5 DWs = 2,560 bytes. This is the limit for ports that do not originate transactions (e.g., Root Complex with peer-to-peer support and Switches).  <b>Infinite units.</b> Initial Credit Value = all 0's for ports that originate transactions (e.g., Root Complex with no peer-to-peer support and Endpoints).
Completion Data (CPLD)	<b>2048 units.</b> Value of the Max_Payload_Size (4096 bytes) including all functions supported by a device (8) divided by the credit size (4 DWs) = 32,768 bytes 2048 credits @ 4 DWs = 32,768 bytes  <b>Infinite units.</b> Initial Credit Value = all 0's for ports that originate transactions (e.g., Root Complex with no peer-to-peer support and Endpoints).

---

## Infinite Credits

Note that a flow control value of 00h will be understood to mean infinite credits during initialization. Following Flow-Control initialization no further advertisements are made. Devices that originate transactions must reserve buffer space for the data or status information that will return during split transactions. These transaction combinations include:

- Non-posted Read requests and return of Completion Data
- Non-posted Read requests and return of Completion Status
- Non-posted Write requests and return of Completion Status

---

## Special Use for Infinite Credit Advertisements.

The specification points out a special consideration for devices that implement only VC0. For example, the only Non-Posted writes are I/O Writes and Configuration Writes both of which are permitted only on VC0. Thus, Non-Posted data buffers are not used for VC1 - VC7 and an infinite value can be advertised for those values. However, the Non-Posted Header must still operate and header credits must still need to be updated.

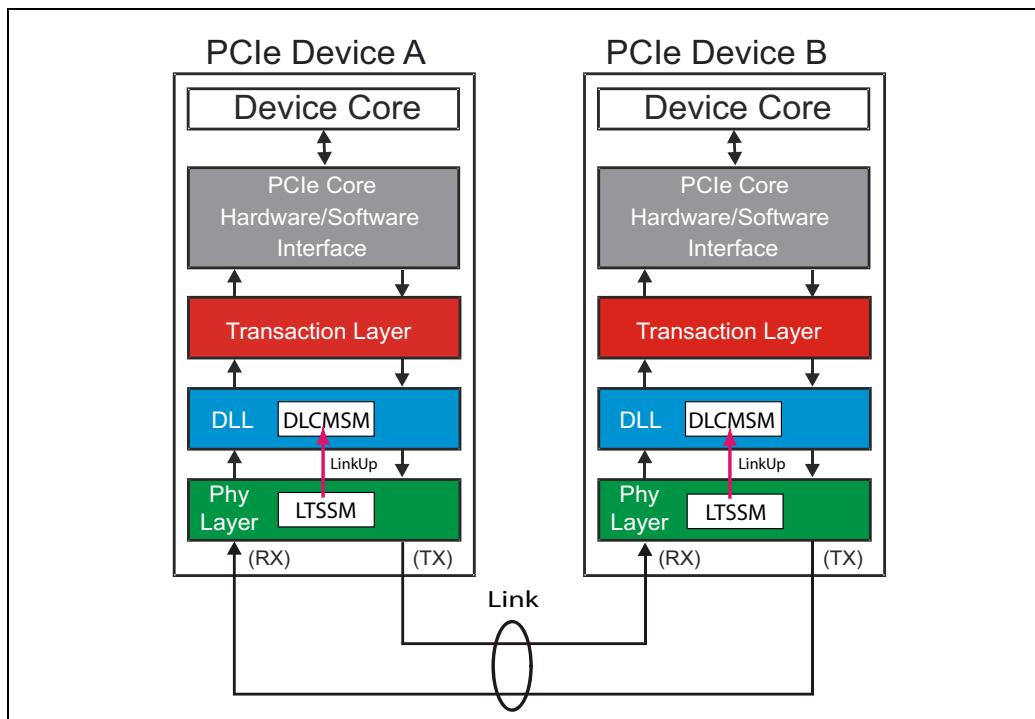
# PCI Express Technology

## Flow Control Initialization

### General

Prior to sending any transactions, flow control initialization is needed. In fact, TLPs cannot be sent across the Link until Flow Control Initialization is performed successfully. Initialization occurs on every Link in the system and involves a handshake between the devices at each end of a link. This process begins as soon as the Physical Layer link training has completed. The Link Layer knows the Physical Layer is ready when it observes the LinkUp signal is active, as illustrated in Figure 6-3.

Figure 6-3: Physical Layer Reports That It's Ready



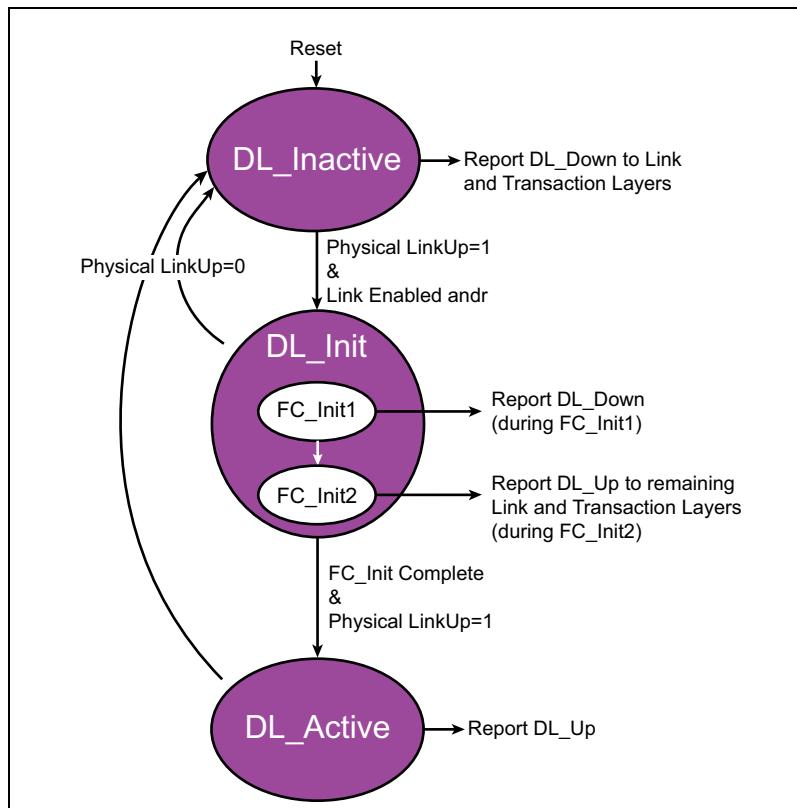
Once started, the Flow Control initialization process is fundamentally the same for all Virtual Channels and is controlled by hardware once a VC has been enabled. VC0 is always enabled by default, so its initialization is automatic.

That allows configuration transactions to traverse the topology and carry out the enumeration process. Other VCs only initialize when configuration software has set up and enabled them at both ends of the Link.

## The FC Initialization Sequence

The flow control initialization process involves the Link Layer's DLCMSM (Data Link Control and Management State Machine). As shown in Figure 6-4 on page 223, a reset puts the state machine into the DL\_Inactive state. While in the DL\_Inactive state, DL\_Down is signaled to both the Link and Transaction Layers. Meanwhile, it waits to see LinkUp from the Physical Layer to indicate that the LTSSM has finished its work and the Physical Layer is ready. That causes a transition to the DL\_Init sub-state, which contains two stages that handle flow control initialization: FC\_INIT1 and FC\_INIT2.

Figure 6-4: The Data Link Control & Management State Machine



## FC\_Init1 Details

During the FC\_INIT1 state, devices continuously send a sequence of 3 InitFC1 Flow Control DLLPs advertising their receiver buffer sizes (see Figure 6-5). According to the spec, the packets must be sent in this order: Posted, Non-posted, and Completions as illustrated in Figure 6-6 on page 225. The specification strongly encourages that these be repeated frequently to make it easier for the receiving device to see them, especially if there are no TLPs or DLLPs to send. Each device should also receive this sequence from its neighbor so it can register the buffer sizes. Once a device has sent its own values and received the complete sequence enough times to be confident that the values were seen correctly, it's ready to exit FC\_INIT1. To do that, it records the received values in its transmit counters, sets an internal flag (FL1), and changes to the FC\_INIT2 state to begin the second initialization step.

Figure 6-5: INIT1 Flow Control DLLP Format and Contents

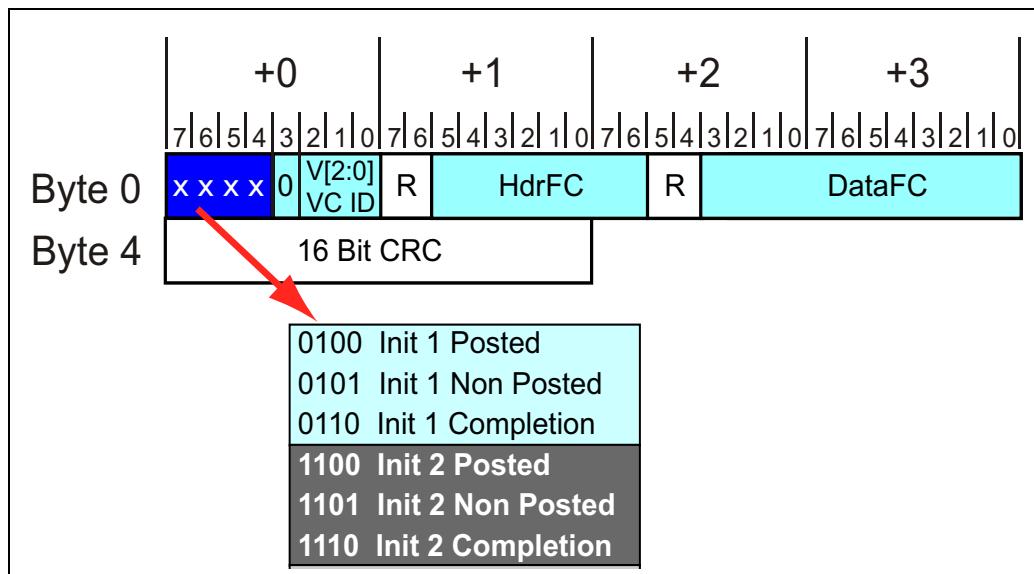
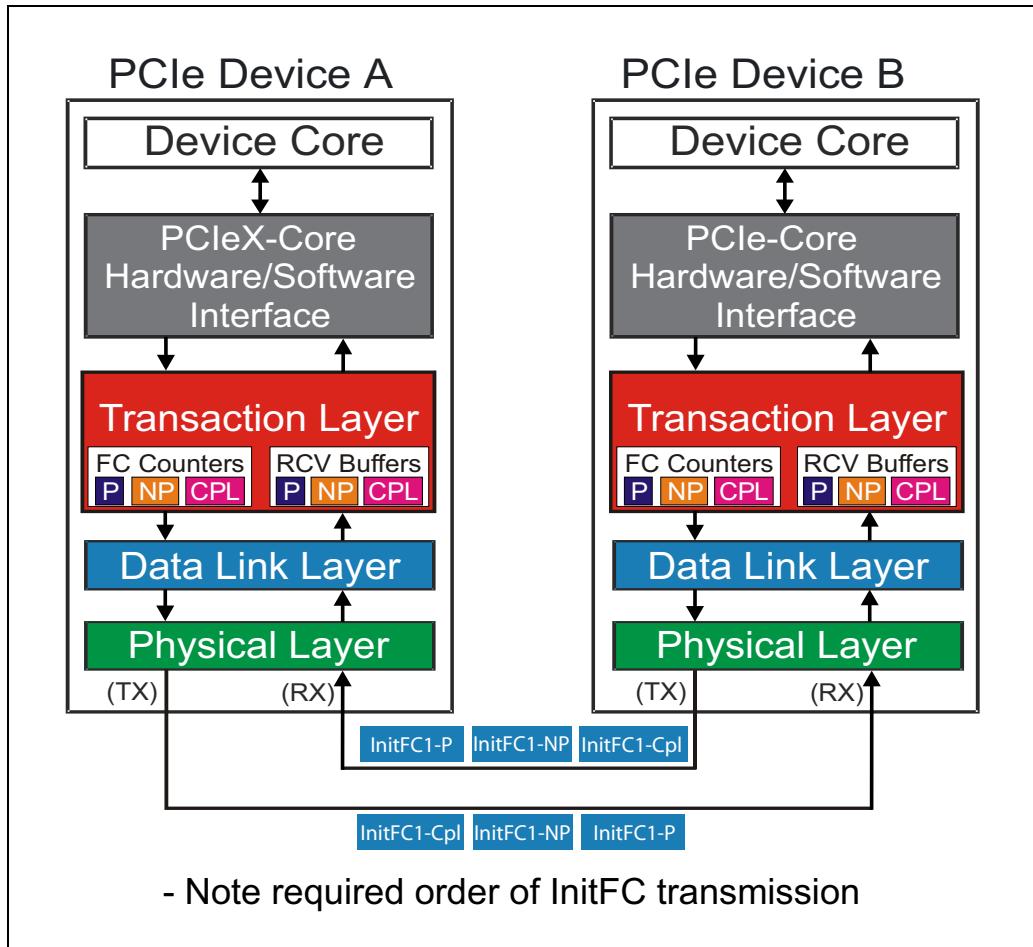


Figure 6-6: Devices Send InitFC1 in the DL\_Init State



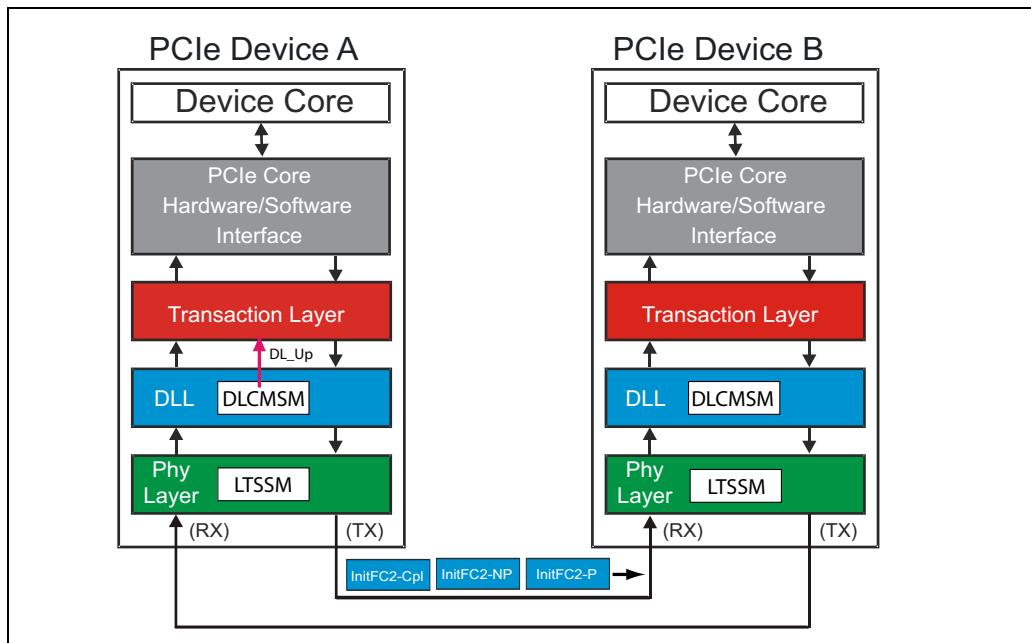
## FC\_Init2 Details

In this state a device continuously sends InitFC2 DLLPs. These are sent in the same sequence as the InitFC1s and contain the same credit information, but they also confirm that FC initialization has succeeded at the sender. Since the device has already registered the values from the neighbor it doesn't need any more credit information and will ignore any incoming InitFC1s while it waits to see InitFC2s. It can even send TLPs at this point, even though initialization hasn't completed for the other side of the Link, and this is indicated to the Transaction Layer by the DL\_Up signal (See Figure 6-7).

# PCI Express Technology

Why is this second initialization step needed? The simple answer is that neighboring devices may finish FC initialization at different times and this method ensures that the late one will continue to receive the FC information it needs even if the neighbor finishes early. Once a device receives an FC\_INIT2 packet for any buffer type, it sets an internal flag (FL2). (It doesn't wait to receive an FC\_Init2 for each type.) Note that FL2 is also set upon receipt of an UpdateFC packet or TLP. When both sides are done and have sent InitFC2s, the DLCMSM transitions to the DL\_Active state and the Link Layer is ready for normal operation.

Figure 6-7: FC Values Registered - Send InitFC2s, Report DL\_Up



## Rate of FC\_INIT1 and FC\_INIT2 Transmission

The specification defines the latency between sending FC\_INIT DLLPs as follows:

- **VC0.** Hardware-initiated flow control of VC0 requires that FC\_INIT1 and FC\_INIT2 packets be transmitted “continuously at the maximum rate possible.” That is, the resend timer is set to a value of zero.
- **VC1-VC7.** When software initiates flow control initialization for other VCs, the FC\_INIT sequence is repeated “when no other TLPs or DLLPs are available for transmission.” However, the latency between the beginning of one sequence to the next can be no greater than 17 $\mu$ s.

---

## Violations of the Flow Control Initialization Protocol

A violation of the flow control initialization protocol can be optionally checked by a device. An error detected can be reported as a Data Link Layer protocol error.

---

## Introduction to the Flow Control Mechanism

---

### General

The specification defines the requirements of the Flow Control mechanism using registers, counters, and mechanisms for reporting, tracking, and calculating whether a transaction can be sent. These elements are not required and the actual implementation is left to the device designer. This section introduces the specification model and serves to explain the concepts and to define the requirements.

---

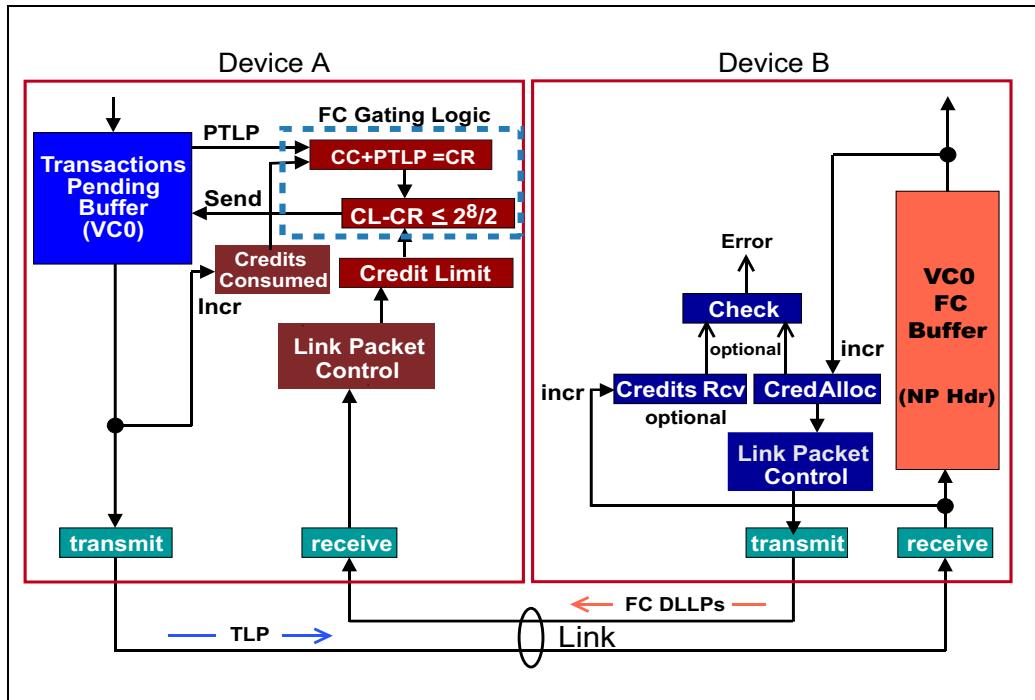
### The Flow Control Elements

Figure 6-8 illustrates the elements used for managing flow control. The diagram shows transactions flowing in a single direction across a Link, and another set of these elements supports transfers in the opposite direction. The primary function of each element is listed below. While these Flow Control elements are duplicated for all six receive buffers, for simplicity this example only deals with non-posted header flow control.

One final element associated with managing flow control is the Flow Control Update DLLP. This is the only Flow Control packet that is used during normal transmission. The format of the FC Update packet is illustrated in Figure 6-9 on page 229.

# PCI Express Technology

Figure 6-8: Flow Control Elements



## Transmitter Elements

- **Transactions Pending Buffer** — holds transactions that are waiting to be sent in the same virtual channel.
- **Credits Consumed counter** — contains the credit sum of all transactions sent for this buffer. This count is abbreviated “CC.”
- **Credit Limit counter** — initialized by the receiver with the size of the corresponding Flow Control buffer. After initialization, Flow Control update packets are sent periodically to update the Flow Control credits as they become available at the receiver. This value is abbreviated “CL.”
- **Flow Control Gating Logic** — performs the calculations to determine if the receiver has sufficient Flow Control credits to accept the pending TLP (PTLP). In essence, this logic checks that the CREDITS\_CONSUMED (CC) plus the credits required for the next Pending TLP (PTLP) does not exceed the CREDIT\_LIMIT (CL). This specification defines the following equation for performing the check, with all values represented in credits.

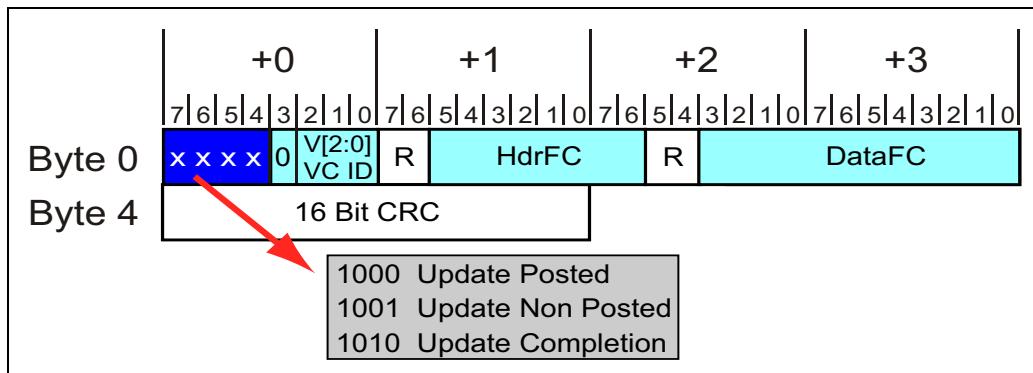
$$CL - (CC + PTLP) \bmod 2^{[FieldSize]} \leq 2^{[FieldSize]} / 2$$

For an example application of this equation, See “Stage 1 — Flow Control Following Initialization” on page 230.

## Receiver Elements

- **Flow Control Buffer** — stores incoming headers or data.
- **Credit Allocated** — tracks the total Flow Control credits that have been allocated (made available). It’s initialized by hardware to reflect the size of the associated Flow Control buffer. The buffer fills as transactions arrive but then they are eventually removed from the buffer by the core logic at the receiver. When they are removed, the number of Flow Control credits is added to the CREDIT\_ALLOCATED counter. Thus the counter tracks the number of credits currently available.
- **Credits Received counter (optional)** — tracks the total credits of all TLPs received into the Flow Control buffer. When flow control is functioning properly, the CREDITS\_RECEIVED count should be equal to or less than the CREDIT\_ALLOCATED count. If this test ever becomes false, a flow control buffer overflow has occurred and an error is detected. The spec recommends that this optional mechanism be implemented and notes that a failure here will be considered a fatal error.

*Figure 6-9: Types and Format of Flow Control DLLPs*



## Flow Control Example

The following example describes the non-posted header Flow Control buffer, and attempts to capture the nuances of the flow control implementation in several situations. The discussion of Flow Control is described with a series of basic stages as follows:

**Stage One** — Immediately following initialization a transaction is transmitted and tracked to explain the basic operation of the counters and registers.

**Stage Two** — The transmitter sends transactions faster than the receiver can process them and the buffer becomes full.

**Stage Three** — When counters roll over to zero, the mechanism still works but there are a couple of issues to consider.

**Stage Four** — The optional receiver error check for a buffer overflow.

---

## Stage 1 — Flow Control Following Initialization

Once flow control initialization has completed, the devices are ready for normal operation. The Flow Control buffer in our example is 2KB in size. We're describing the non-posted header buffer, and each credit is 5 dwords in size or 20 bytes. That means 102d (66h) Flow Control units are available. Figure 6-10 on page 231 illustrates the elements involved, including the values that would be in each counter and register following flow control initialization.

When the transmitter is ready to send a TLP, it must first check Flow Control credits. Our example is simple because a non-posted header is the only packet being sent and it always requires just one Flow Control credit, and we are also assuming that no data is included in the transaction.

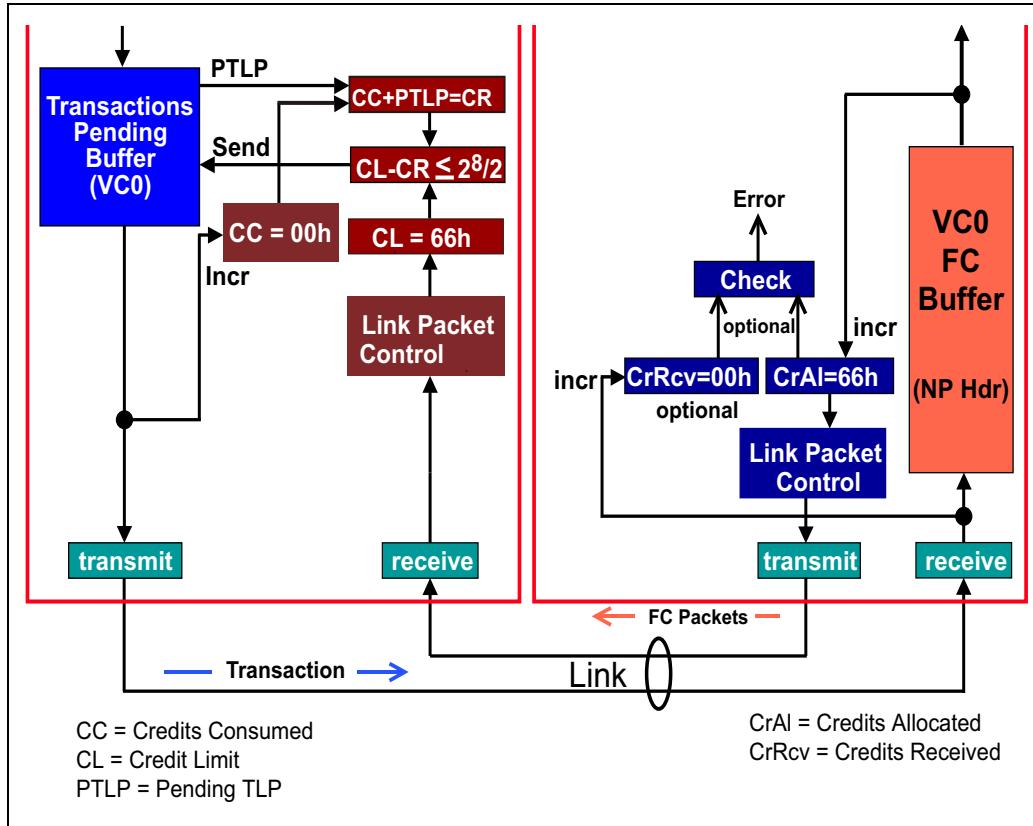
The header credit check is made using unsigned arithmetic (2's complement), and must satisfy the following formula:

$$CL - (CC + PTLP) \bmod 2^{[FieldSize]} \leq 2^{[FieldSize]}/2$$

Substituting values from Figure 6-10 yields:

$$\begin{aligned} 66h - (00h + 01h) \bmod 2^8 &\leq 2^8/2 \\ 66h - 01h \bmod 256 &\leq 80h \end{aligned}$$

Figure 6-10: Flow Control Elements Following Initialization



In this case, the current CREDITS\_CONSUMED count (CC) is added to the PTLP credits required, to determine the CREDITS\_REQUIRED (CR), and that gives  $00h + 01h = 01h$ . The CREDITS\_REQUIRED count is subtracted from the CREDIT\_LIMIT count (CL) to determine whether or not sufficient credits are available.

The following description incorporates a brief review of 2's complement subtraction. When performing subtraction using 2's complement the number to be subtracted is complemented (1's complement) and 1 is added (2's complement). This value is then added to the number from which we wish to subtract. Any carry due to the addition is dropped.

# PCI Express Technology

Credit Check:

$$CL \ 01100110b \ (66h) - CR \ 00000001b \ (01h) = n$$

CR is converted to 2's complement:

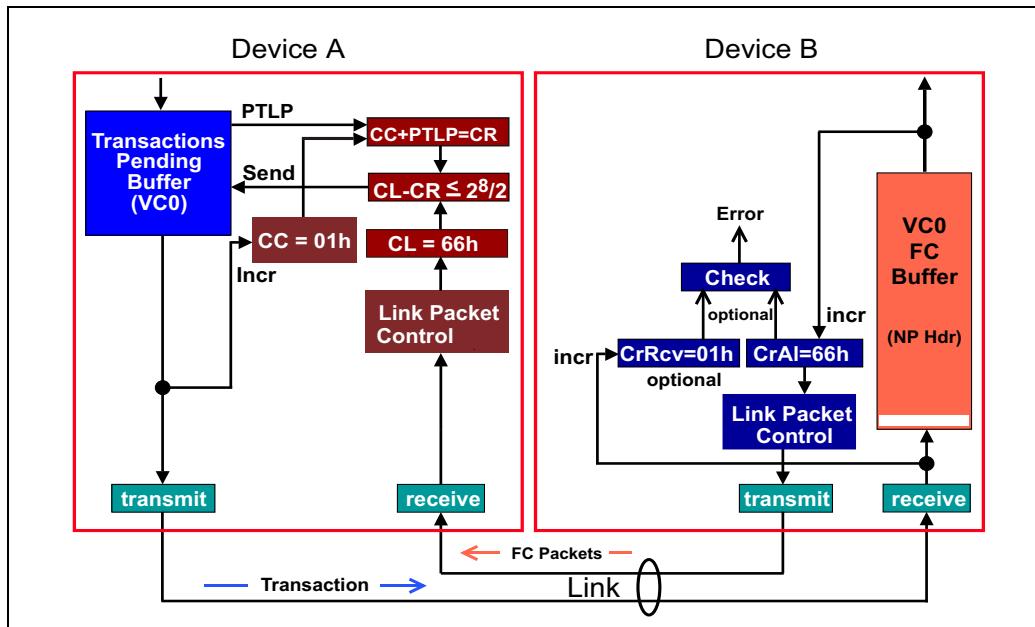
$$\begin{aligned} &00000001b \ (\text{CR}) \\ &11111110b \ (\text{CR inverted}) \\ &11111110b +1 \\ &11111111b \ (2\text{'s complement}) \end{aligned}$$

2's complement added to CL:

$$\begin{aligned} &01100110 \ (\text{CL}) \\ &\underline{11111111} \ (2\text{'s complement of CR}) \\ &01100101 = 65h \ (\text{carry bit is dropped}) \end{aligned}$$

Is result  $\leq 80h$ ? Yes. If the subtraction result is equal to or less than half the max value, which is tracked with a modulo 256 counter (128), then we know there is sufficient space in the receiver buffer and this packet can be sent. The decision to use only half the counter value avoids a potential count alias problem. See "Stage 3 — Counters Roll Over" on page 234.

Figure 6-11: Flow Control Elements After First TLP Sent



### Stage 2 — Flow Control Buffer Fills Up

Assume now that the receiver has been unable to remove transactions from the Flow Control buffer for some time. Perhaps the device core logic was temporarily busy and unable to process transactions. Eventually, the Flow Control buffer becomes completely full, as shown in Figure 6-12 on page 234. If the transmitter wishes to send another TLP and checks the Flow Control credits:

Credit Limit (CL)= 66h

Credits Required (CR) = 67h

CL 01100110 (66)

CR 10011001 (add 2's complement of 67h)

11111111 = FFh<=80h (not true; don't send packet)

This channel is blocked until an Update Flow Control DLLP is received with a new CREDIT\_LIMIT value of 67h or greater. When the new value is loaded into the CL register the transmitter credit check will pass the test and a TLP can be sent.

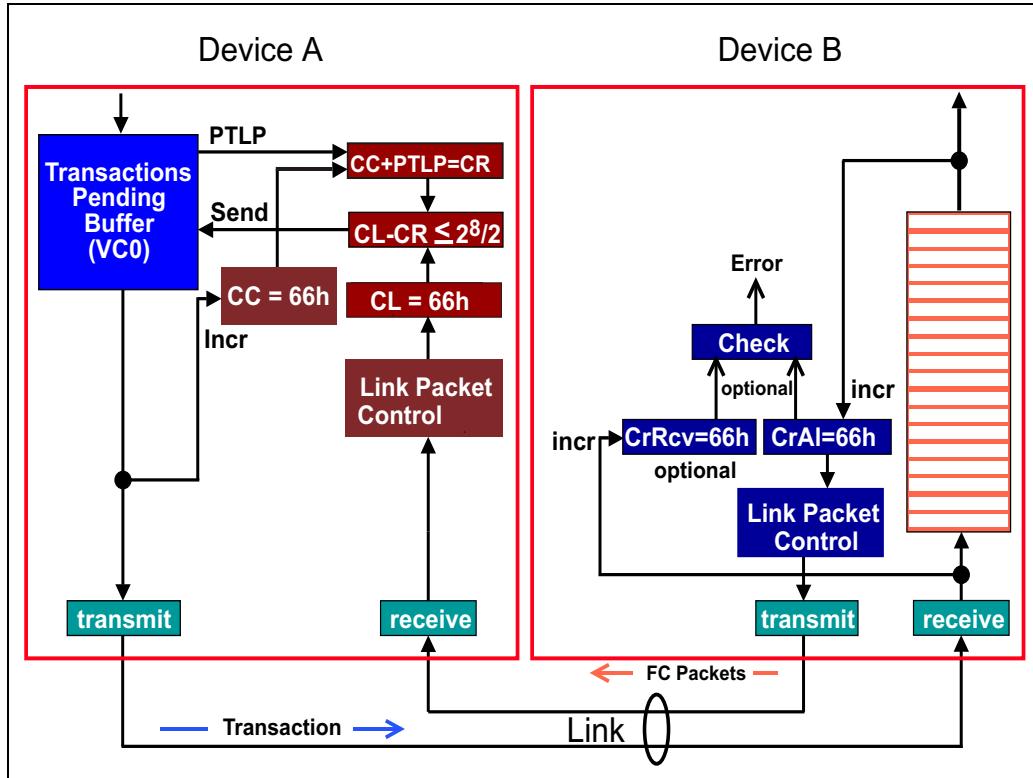
CL 01100111 (67)

CR 10011001 add 2's complement of 67

00000000 = 00h<=80h (true, send transaction)

# PCI Express Technology

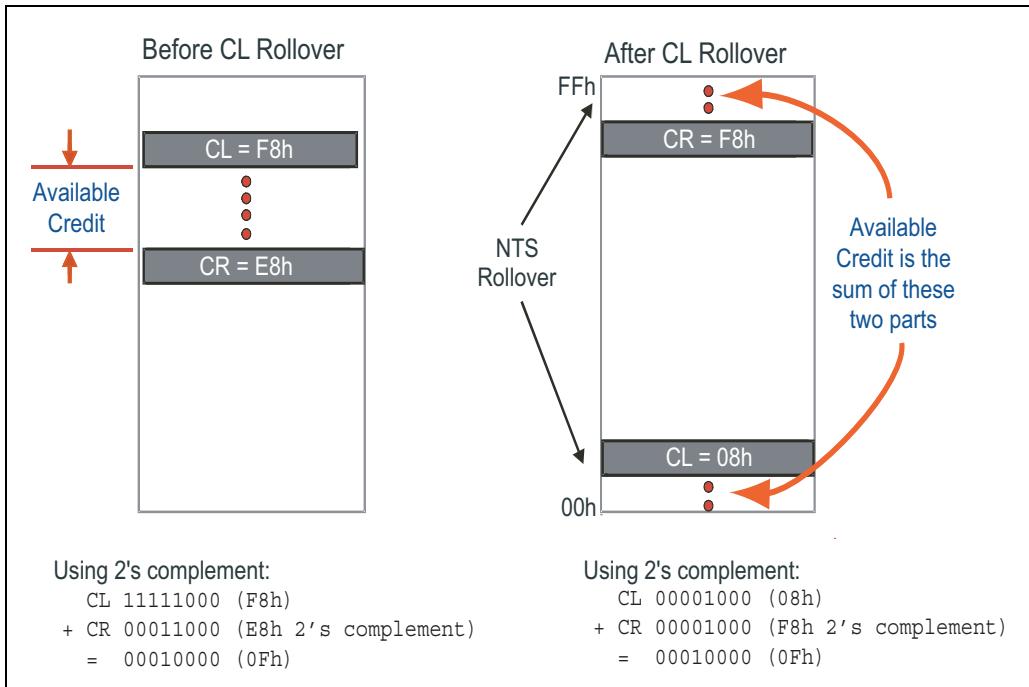
Figure 6-12: Flow Control Elements with Flow Control Buffer Filled



## Stage 3 — Counters Roll Over

Since the Credit Limit (CL) and Credits Required (CR) counts only increment upward, they eventually roll over back to zero. When CL rolls over and CR has not, the credit check (CL-CR) results in a small CL value and a large CR value. However, what might appear to be a problem is not when using unsigned arithmetic. As described in the previous examples the results are handled correctly when performing 2's complement subtraction. Figure 6-13 on page 235 shows the CL and CR counts before and after CL rollover along with the 2's complement results.

Figure 6-13: Flow Control Rollover Problem



## Stage 4 — FC Buffer Overflow Error Check

Although it's optional to do so, the specification recommends implementation of the FC buffer overflow error checking mechanism. Figure 6-14 on page 236 shows the elements associated with the overflow error check that include:

- Credits Received (CR) counter
- Credits Allocated (CA) counter
- Error Check Logic

This permits the receiver to track Flow Control credits in the same manner as the transmitter. If flow control is working correctly, the transmitter's Credits Consumed count will never exceed its Credit Limit, and the receiver's Credits Received count will never exceed its Credits Allocated count.

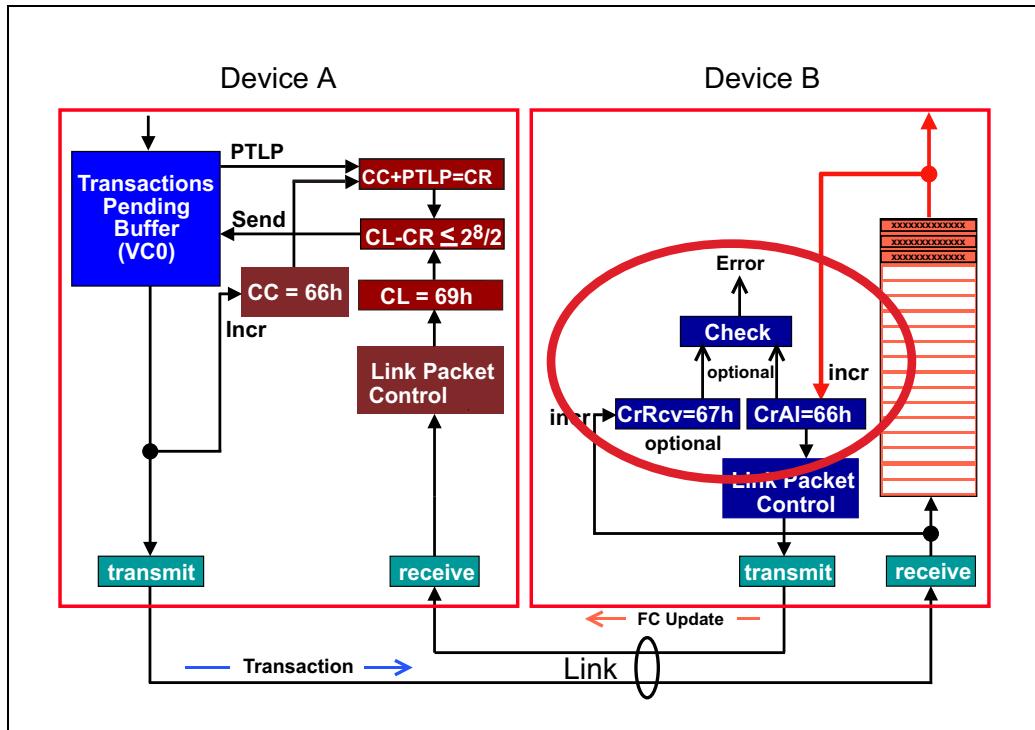
# PCI Express Technology

An overflow condition is detected if the following formula evaluates true. Note that the field size is either 8 (headers) or 12 (data):

$$(CA - CR) \bmod 2^{[FieldSize]} > 2^{[FieldSize]}/2$$

If it does evaluate true, then more credits have been sent to the FC buffer than were available and an overflow has occurred. Note that the 1.0a version of the specification defines the equation as  $\geq$  rather than  $>$  as shown above. That appears to be an error, because when CA = CR no overflow condition exists.

Figure 6-14: Buffer Overflow Error Check



## Flow Control Updates

The receiver must regularly update its neighboring device with Flow Control credits that become available when transactions are removed from the buffer. Figure 6-15 on page 238 illustrates an example where the transmitter was previously blocked from sending header transactions because the buffer was full. In the illustration, the receiver has just removed three headers from the Flow Control buffer. More space is now available, but the neighboring device is unaware of this. As headers are removed from the buffer, the CREDITS\_ALLOCATED count increments from 66h to 69h. This new count is reported to the CREDIT\_LIMIT register of the neighboring device using a Flow Control update packet. Once the credit limit has been updated, transmission of additional TLPs can proceed.

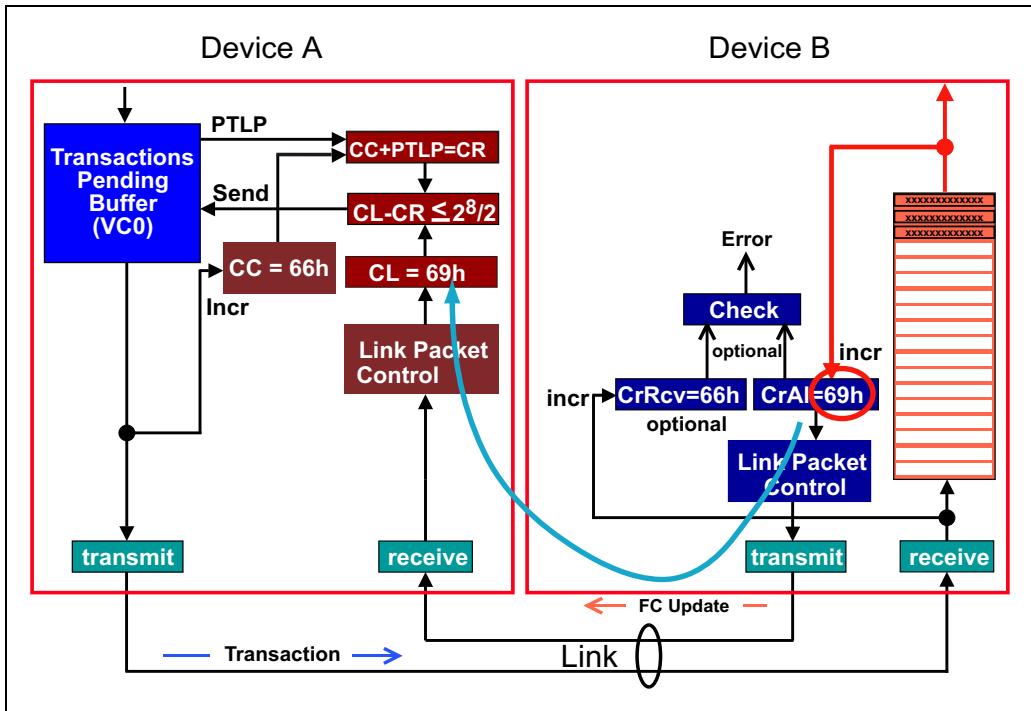
An interesting note here is that the update reports the actual value of the Credits Allocated register. It would have worked to report just the change in the register, as perhaps “+3 credits on NP Headers” for example, but that represents a potential problem. To understand the risk, consider what would happen if the DLLP containing that increment information was lost for some reason. There is no replay mechanism for DLLPs; if an error occurs the packet is simply dropped. In this case, the increment information would be lost without a means of recovering it.

If, on the other hand, the actual value of the register is reported instead and the DLLP fails, the next DLLP that succeeds will get the counters back in synchronization. In that case some time might be wasted if the transmitter is waiting on the FC credits before it can send the next TLP, but no information is lost.

# PCI Express Technology

---

Figure 6-15: Flow Control Update Example

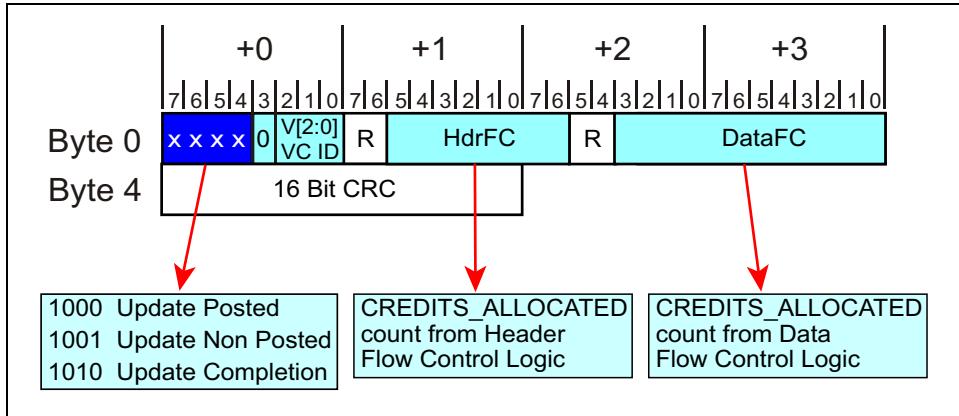


---

## FC\_Update DLLP Format and Content

Recall that Flow Control update packets, like the Flow Control initialization packets, contain two credit fields, one for header and one for data, as shown in Figure 6-16 on page 239. The receiver's credit values reported in the HdrFC and DataFC fields may have been updated many times or not at all since the last update packet was sent.

Figure 6-16: Update Flow Control Packet Format and Contents



## Flow Control Update Frequency

The specification defines a variety of rules and suggested implementations that govern when and how often Flow Control Update DLLPs should be sent. These are motivated by a desire to:

- Notify the transmitting device as early as possible about new credits allocated, especially if any transactions were previously blocked.
- Establish worst-case latency between FC Packets.
- Balance the requirements associated with flow control operation, such as:
  - the need to report credits often enough to prevent transaction blocking
  - the desire to reduce the Link bandwidth needed for FC\_Update DLLPs
  - selecting the optimum buffer size
  - selecting the maximum data payload size
- Detect violations of the maximum latency between Flow Control packets.

Flow Control updates are permitted only when the Link is in the active state (L0 or L0s). All other Link states represent more aggressive power management that have longer recovery latencies.

## Immediate Notification of Credits Allocated

When a Flow Control buffer is so full that maximum-sized packets cannot be sent, the spec requires immediate delivery of a FC\_Update DLLP when more space becomes available. Two cases exist:

# PCI Express Technology

---

- **Maximum Packet Size = 1 Credit.** When packet transmission is blocked due to a buffer full condition for non-infinite NPH, NPD, PH, and CPLH buffer types, an UpdateFC packet must be scheduled for Transmission when one or more credits are made available (allocated) for that buffer type.
- **Maximum Packet Size = Max\_Payload\_Size.** Flow Control buffer space may decrease to the extent that a maximum-sized packet cannot be sent for non-infinite PD and CPLD credit types. In this case, when one or more additional credits are allocated, an Update FCP must be scheduled for transmission.

## Maximum Latency Between Update Flow Control DLLPs

The transmission frequency of Update FCPs for each FC credit type (non-infinite) must be scheduled for transmission at least once every 30 µs (-0%/+50%). If the Extended Sync bit within the Control Link register is set, updates must be scheduled no later than every 120 µs (-0%/+50%). Note that Update FCPs may be scheduled for transmission more frequently than is required.

## Calculating Update Frequency Based on Payload Size and Link Width

The specification offers a formula for calculating the frequency at which update packets need to be sent for maximum data payload sizes and Link widths. The formula, shown below, defines FC Update delivery intervals in symbol times. For reference, a symbol time is defined as the time it takes to deliver one symbol: 4ns for Gen1, 2ns for Gen2, 1ns for Gen3. Table 6-3, Table 6-4 and Table 6-5 show the unadjusted FC Update values for each speed.

$$\frac{(MaxPayloadSize + TLPOverhead) \times UpdateFactor}{LinkWidth} + InternalDelay$$

- **MaxPayloadSize** = The value in the Max\_Payload\_Size field of the Device Control register
- **TLPOverhead** = the constant value (28 symbols) representing the additional TLP components that consume Link bandwidth (TLP Prefix, Sequence Number, Packet Header, LCRC, Framing Symbols)
- **UpdateFactor** = the number of maximum size TLPs sent during the interval between UpdateFC Packets received. This number is intended to balance Link bandwidth efficiency and receive buffer sizes – the value varies with Max\_Payload\_Size and Link width

## Chapter 6: Flow Control

---

- **LinkWidth** = The number of Lanes the Link is using
- **InternalDelay** = a constant value of 19 symbol times that represents the internal processing delays for received TLPs and transmitted DLLPs

The relationship defined by the formula shows that the frequency of update packet delivery decreases as the Linkwidth increases and suggests a timer that triggers scheduling of update packets. Note that this formula does not account for delays associated with the receiver or transmitter being in the L0s power management state.

*Table 6-3: Gen1 Unadjusted AckNak\_LATENCY\_TIMER Values (Symbol Times)*

Max Payload	x1 Link	x2 Link	x4 Link	x8 Link	x12 Link	x16 Link	x32 Link
128 Bytes	237 UF=1.4	128 UF=1.4	73 UF=1.4	67 UF=2.5	58 UF=3.0	48 UF=3.0	33 UF=3.0
256 Bytes	416 FC=1.4	217 FC=1.4	118 UF=1.4	107 UF=2.5	90 UF=3.0	72 UF=3.0	45 UF=3.0
512 Bytes	559 UF=1.0	289 UF=1.0	154 UF=1.0	86 UF=1.0	109 UF=2.0	86 UF=2.0	52 UF=2.0
1024 Bytes	1071 UF=1.0	545 UF=1.0	282 UF=1.0	150 UF=1.0	194 UF=2.0	150 UF=2.0	84 UF=2.0
2048 Bytes	2095 UF=1.0	1057 UF=1.0	538 UF=1.0	278 UF=1.0	365 UF=2.0	278 UF=2.0	148 UF=2.0
4096 Bytes	4143 UF=1.0	2081 UF=1.0	1050 UF=1.0	534 UF=1.0	706 UF=2.0	534 UF=2.0	276 UF=2.0

*Table 6-4: Gen2 Unadjusted AckNak\_LATENCY\_TIMER Values (Symbol Times)*

Max Payload	x1 Link	x2 Link	x4 Link	x8 Link	x12 Link	x16 Link	x32 Link
128 Bytes	288 UF=1.4	179 UF=1.4	124 UF=1.4	118 UF=2.5	109 UF=3.0	99 UF=3.0	84 UF=3.0
256 Bytes	467 FC=1.4	268 FC=1.4	169 UF=1.4	158 UF=2.5	141 UF=3.0	123 UF=3.0	96 UF=3.0

# PCI Express Technology

---

Table 6-4: Gen2 Unadjusted AckNak\_LATENCY\_TIMER Values (Symbol Times) (Continued)

Max Payload	x1 Link	x2 Link	x4 Link	x8 Link	x12 Link	x16 Link	x32 Link
512 Bytes	610 UF=1.0	340 UF=1.0	205 UF=1.0	137 UF=1.0	160 UF=2.0	137 UF=2.0	103 UF=2.0
1024 Bytes	1122 UF=1.0	596 UF=1.0	333 UF=1.0	201 UF=1.0	245 UF=2.0	201 UF=2.0	135 UF=2.0
2048 Bytes	2146 UF=1.0	1108 UF=1.0	589 UF=1.0	329 UF=1.0	416 UF=2.0	329 UF=2.0	199 UF=2.0
4096 Bytes	4194 UF=1.0	2132 UF=1.0	1101 UF=1.0	585 UF=1.0	757 UF=2.0	585 UF=2.0	327 UF=2.0

Table 6-5: Gen3 Unadjusted AckNak\_LATENCY\_TIMER Values (Symbol Times)

Max Payload	x1 Link	x2 Link	x4 Link	x8 Link	x12 Link	x16 Link	x32 Link
128 Bytes	333 UF=1.4	224 UF=1.4	169 UF=1.4	163 UF=2.5	154 UF=3.0	144 UF=3.0	129 UF=3.0
256 Bytes	512 FC=1.4	313 FC=1.4	214 UF=1.4	203 UF=2.5	186 UF=3.0	168 UF=3.0	141 UF=3.0
512 Bytes	655 UF=1.0	385 UF=1.0	250 UF=1.0	182 UF=1.0	205 UF=2.0	182 UF=2.0	148 UF=2.0
1024 Bytes	1167 UF=1.0	641 UF=1.0	378 UF=1.0	246 UF=1.0	290 UF=2.0	246 UF=2.0	180 UF=2.0
2048 Bytes	2191 UF=1.0	1153 UF=1.0	643 UF=1.0	374 UF=1.0	461 UF=2.0	374 UF=2.0	244 UF=2.0
4096 Bytes	4239 UF=1.0	2177 UF=1.0	1146 UF=1.0	630 UF=1.0	802 UF=2.0	630 UF=2.0	372 UF=2.0

The specification recognizes that the formula will be inadequate for many applications such as those that stream large blocks of data. These applications may require buffer sizes larger than the minimum specified, as well as a more sophisticated update policy in order to optimize performance and reduce

power consumption. Because a given solution is dependent on the particular requirements of an application, no definition for such policies is provided.

---

## Error Detection Timer — A Pseudo Requirement

The specification defines an optional time-out mechanism for Flow Control packets that is highly recommended and may become a requirement in future versions of the specification. The maximum latency between FC packets for a given credit type is 120 $\mu$ s, and this timeout has a maximum limit of 200 $\mu$ s. A separate timer is implemented for each FC credit type (P, NP, Cpl), and each timer is reset when a FC Update DLLP of the corresponding type is received. Note that a timer associated with infinite FC credit values must not report an error.

Apart from the infinite case, a timeout implies a serious problem with the Link. If it occurs, the Physical Layer is signaled to go into the Recovery state and retrain the Link in hopes of clearing the error condition. Timer characteristics include:

- Operates only when the Link is in an active state (L0 or L0s).
- Max time limited to 200  $\mu$ s (-0%/+50%)
- Timer is reset when any Init or Update FCP is received, or optionally by receipt of any DLLP.
- Timeout forces the Physical Layer to enter Link Training and Status State Machine (LTSSM) Recovery state.

## **PCI Express Technology**

---

---

---

# 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

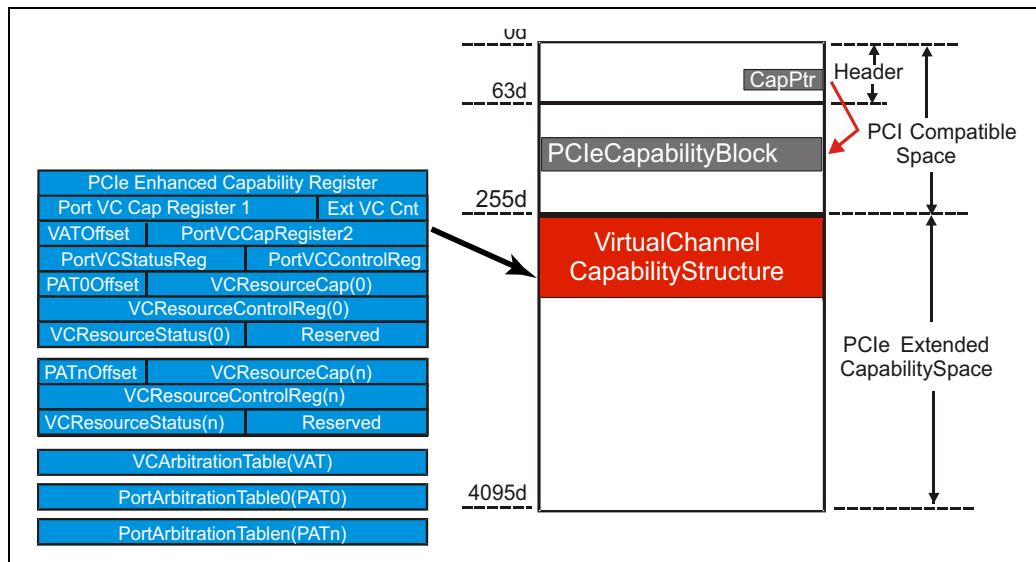
# PCI Express Technology

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.

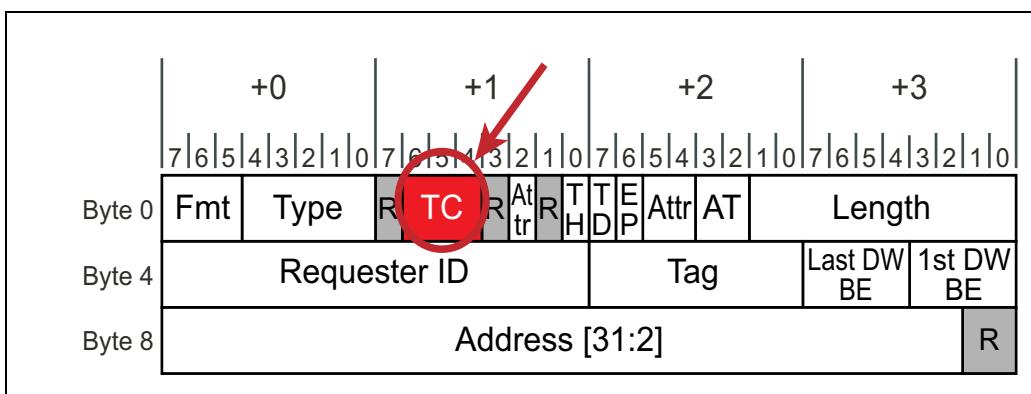
Figure 7-1: Virtual Channel Capability Registers



## 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.

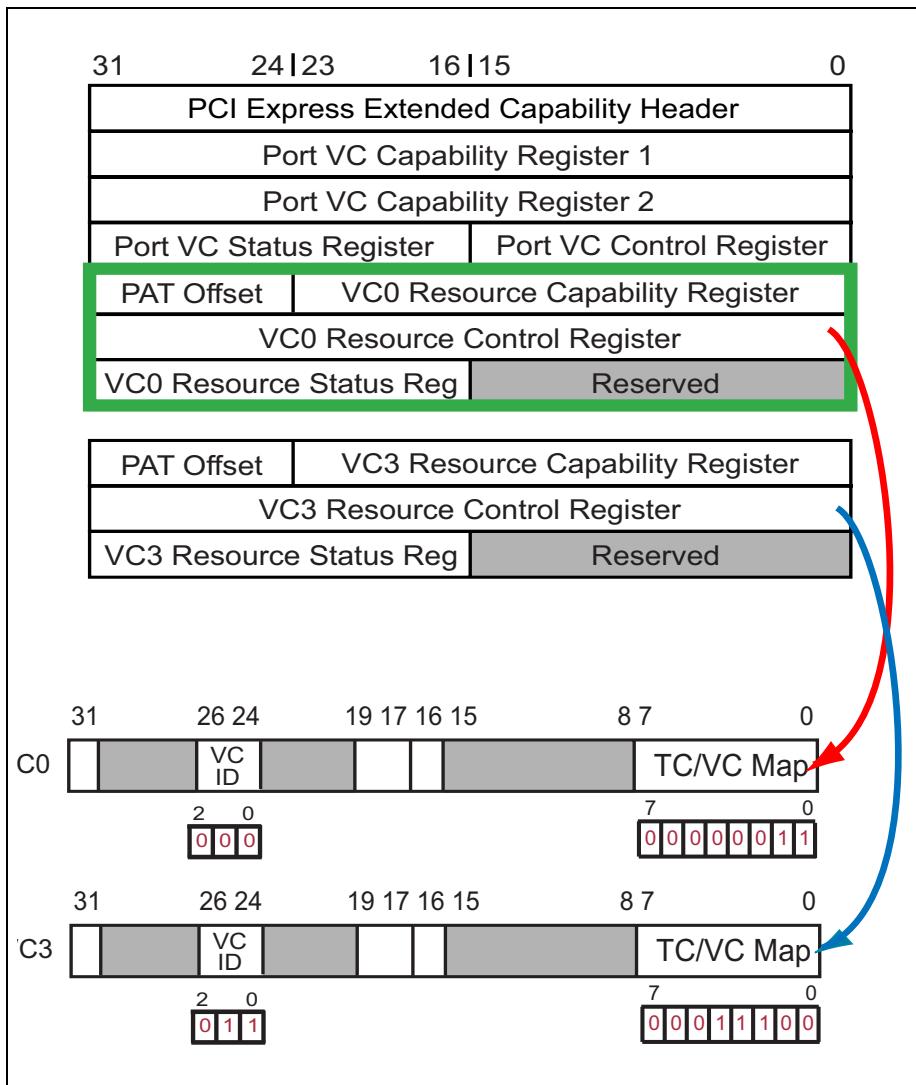
Software has a great deal of flexibility in assigning VC IDs and mapping the TCs, but there are some rules regarding the TC/VC mapping:

- 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.

## Chapter 7: Quality of Service

Figure 7-3: TC to VC Mapping Example



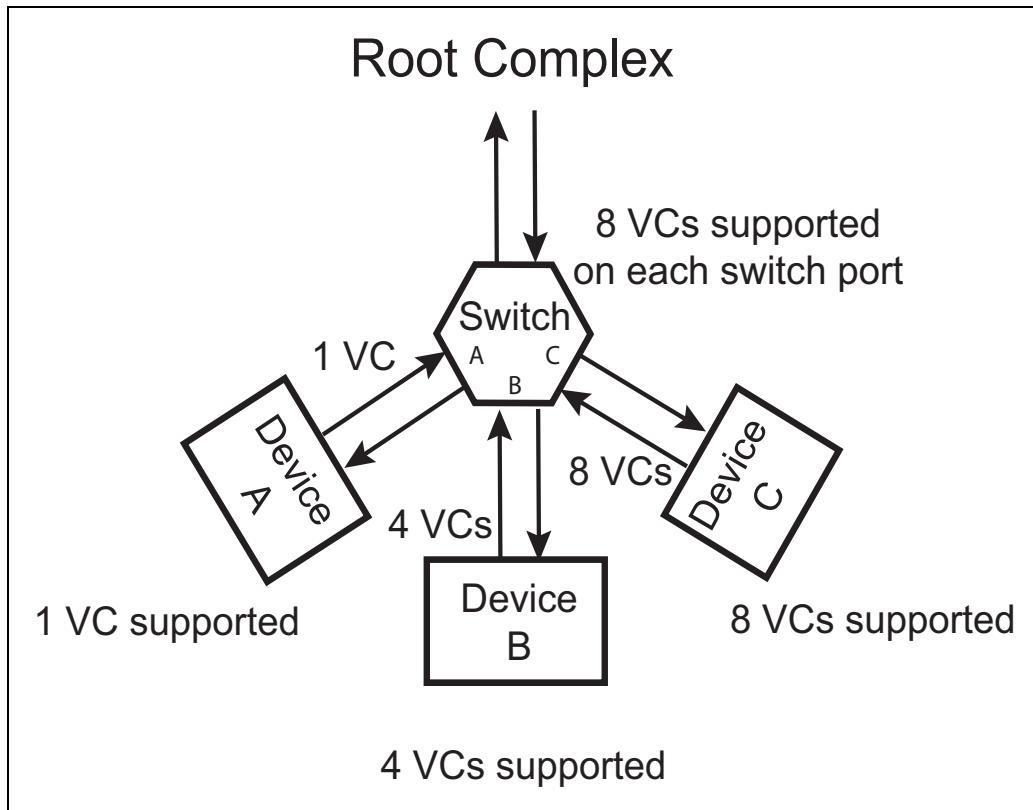
### Determining the Number of VCs to be Used

Software checks the number of VCs supported by the devices attached to a common link and would usually assign the greatest number of VCs that both devices can support. Consider the example topology in Figure 7-4 on page 250.

## PCI Express Technology

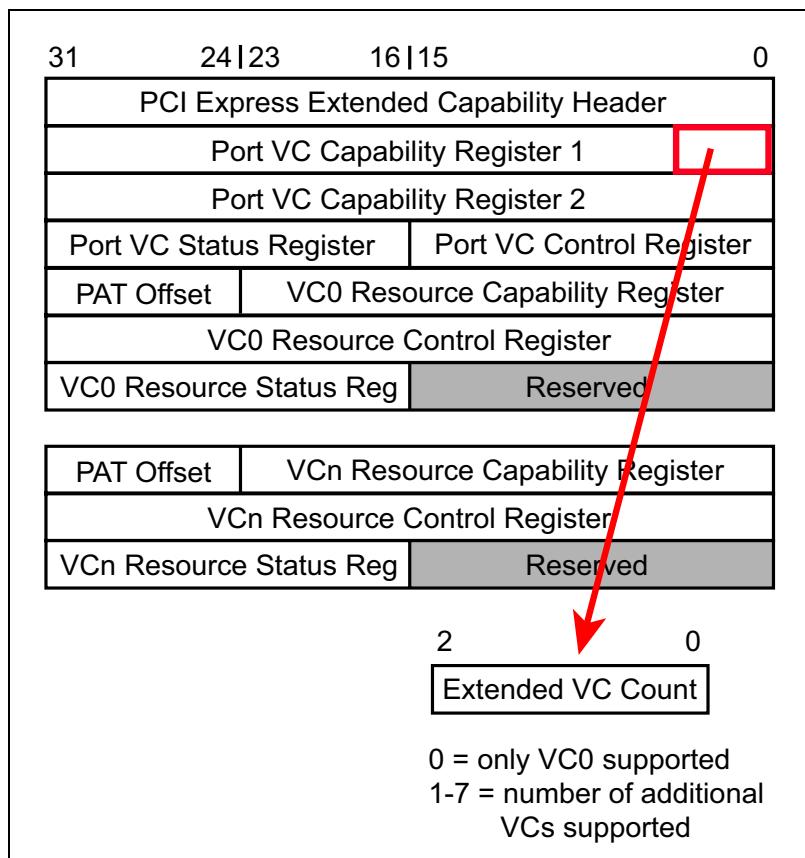
Here, the switch supports all 8 VCs on each of its ports, while Device A supports only the default VC0, Device B supports 4 VCs, and Device C supports 8 VCs. Note that even though switch port A supports all 8 VCs, Device A only supports VC0, so 7 VCs are left unused in switch port A. Similarly, only 4 VCs are used by switch port B.

Figure 7-4: Multiple VCs Supported by a Device



Configuration software determines the maximum number of VCs supported by each port interface by reading the *Extended VC Count* field in the Virtual Channel Capability registers, as shown in Figure 7-5 on page 251. Software checks the Extended VC Count at both ends of the Link and selects the highest common count. Using all the available VCs is not mandatory, though. Software may choose to enable fewer VCs as well.

Figure 7-5: Extended VCs Supported Field



## Assigning VC Numbers (IDs)

Configuration software assigns a number (ID) to each of the VCs, except VC0 which is always hardwired. As shown in Figure 7-3 on page 249, the VC Capabilities registers include 12 bytes of configuration registers for each VC. The first set of registers always applies to VC0. The *Extended VC Count* field defines the number of additional VCs implemented by this port, each of which will have a set of registers. The value “n” represents the number of additional VCs implemented. For example, if the *Extended VC Count* contains a value of 3, then there are three VCs and register sets in addition to VC0.

Software assigns a number for each of the additional VCs via the *VC ID* field. (See Figure 7-3 on page 249) The IDs don't have to be contiguous but each number can only be used once.

---

## VC Arbitration

---

### General

If a device has more than one VC and they all have a packet ready to send, VC arbitration determines the order of packet transmission. Any of several schemes can be chosen by software from among the options implemented by hardware. The goals are to implement the desired service policy and ensure that all transactions are making forward progress to prevent inadvertent time-outs. In addition, VC Arbitration is affected by the requirements associated with flow control and transaction ordering. These topics are discussed in other chapters, but they affect arbitration, too, because:

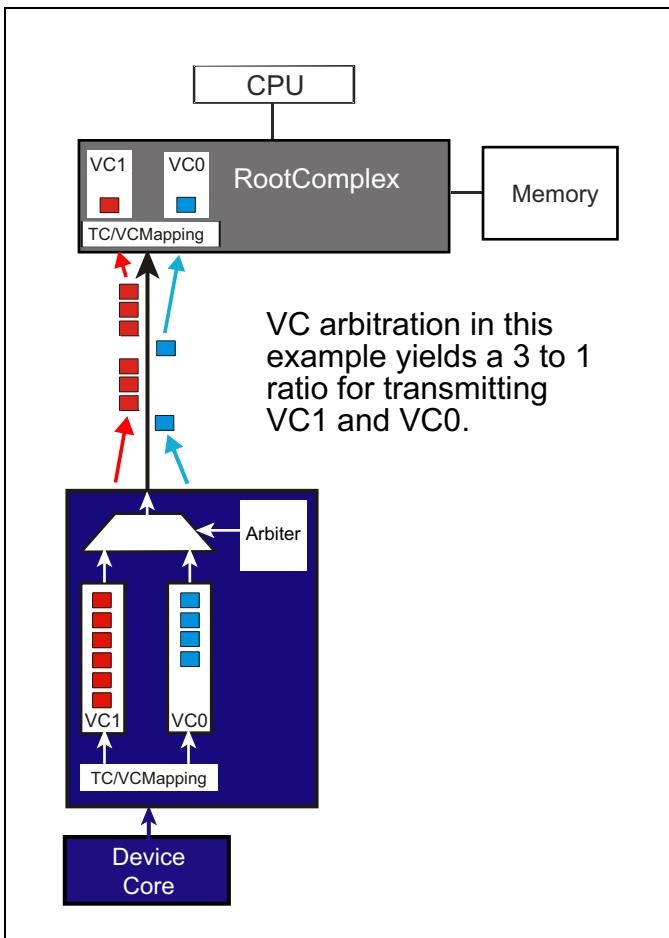
- Each supported VC provides its own buffers and flow control.
- Transactions mapped to the same VC are normally passed along in strict order (although there are exceptions, such as when a packet has the "Relaxed Ordering" attribute bit set).
- Transaction ordering only applies within a VC, so there's no ordering relationship among packets assigned to different VCs.

The example in Figure 7-6 on page 253 illustrates two VCs (VC0 and VC1) with a transmission priority based on a 3:1 ratio, meaning three VC1 packets are sent for every one VC0 packet. The device core sends requests (including a TC value) to the TC/VC Mapping logic. Based on the programmed mapping, the packet is placed into the appropriate VC buffer for transmission. Finally, the VC arbiter determines the VC priority for forwarding the packets. This example illustrates the flow in one direction, but the same logic exists for transmitting in the opposite direction at the same time.

The VC capability registers provide three basic VC arbitration approaches:

1. Strict Priority Arbitration — the highest numbered VC with a packet ready always wins.
2. Group Arbitration — VCs are divided by hardware into one low-priority group and one high-priority group. The low-priority group uses an arbitration method selected by software from the available choices, while the high-priority group always uses strict-priority arbitration.
3. Hardware Fixed arbitration — scheme built into the hardware.

Figure 7-6: VC Arbitration Example



---

## Strict Priority VC Arbitration

The default priority scheme is based on the inherent priority of VC IDs (VC0=lowest priority and VC7=highest priority). The mechanism is automatic and requires no configuration. Figure 7-7 on page 254 illustrates a strict priority arbitration example that includes all VCs. The VC ID governs the order in which transactions are sent. The maximum number of VCs that use strict priority arbitration cannot be greater than the value in the *Extended VC Count* field.

# PCI Express Technology

---

(See Figure 7-5 on page 251.) Furthermore, if the designer has chosen strict priority arbitration for all VCs supported, the *Low Priority Extended VC Count* field of Port VC Capability Register 1 is hardwired to zero. (See Figure 7-8 on page 255.

Figure 7-7: Strict Priority Arbitration

VC Resources	Priority Order
8th VC	VC7 Highest
7th VC	VC6
6th VC	VC5
5th VC	VC4
4th VC	VC3
3rd VC	VC2
2nd VC	VC1
1st VC	VC0 Lowest

Strict priority requires that higher-numbered VCs always get precedence over lower-priority VCs. For example, if all eight VCs are governed by strict priority, then packets in VC0 can only be sent when no other VCs have packets pending. This achieves the goal of giving the highest priority packets very high bandwidth with minimal latencies. However, strict priority has the potential to starve low-priority channels for bandwidth, so care must be taken to ensure this doesn't happen. The spec requires that high priority traffic be regulated to avoid starvation, and gives two possible methods of regulation:

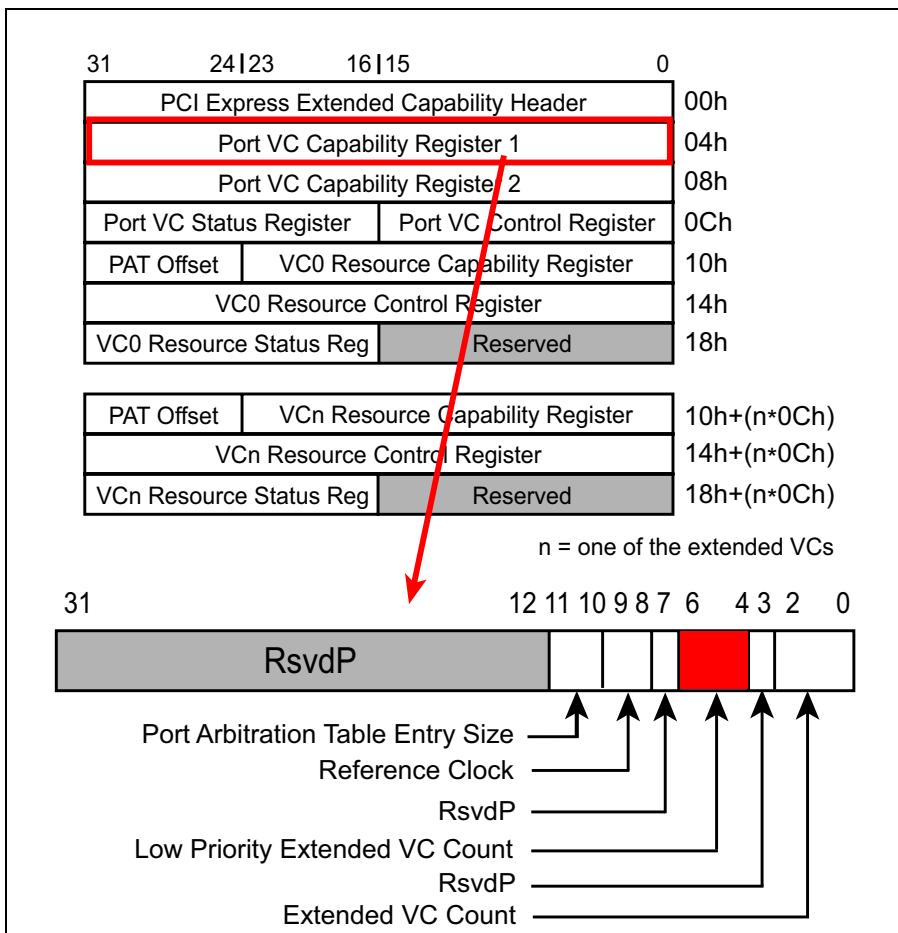
- The originating port can restrict the injection rate of high priority packets to allow more bandwidth for lower priority transactions.
- Switches can regulate multiple traffic flows at the egress port. This method may limit the throughput from high bandwidth applications and devices that attempt to exceed the limitations of the available bandwidth.

A device designer may also limit the number of VCs that participate in strict priority by splitting the VCs into a low-priority group and a high-priority group as discussed in the next section.

## Group Arbitration

Figure 7-8 illustrates the *Low Priority Extended VC Count* field within VC Capability Register 1. This read-only field specifies a VC ID that identifies the upper limit of the low-priority arbitration group for this device. For example, if this value is 4, then VC0-VC4 are members of the low-priority group and VC5-VC7 are in the high-priority group. Note that a *Low Priority Extended VC Count* of 7 means that no strict priority is used.

*Figure 7-8: Low-Priority Extended VCs*

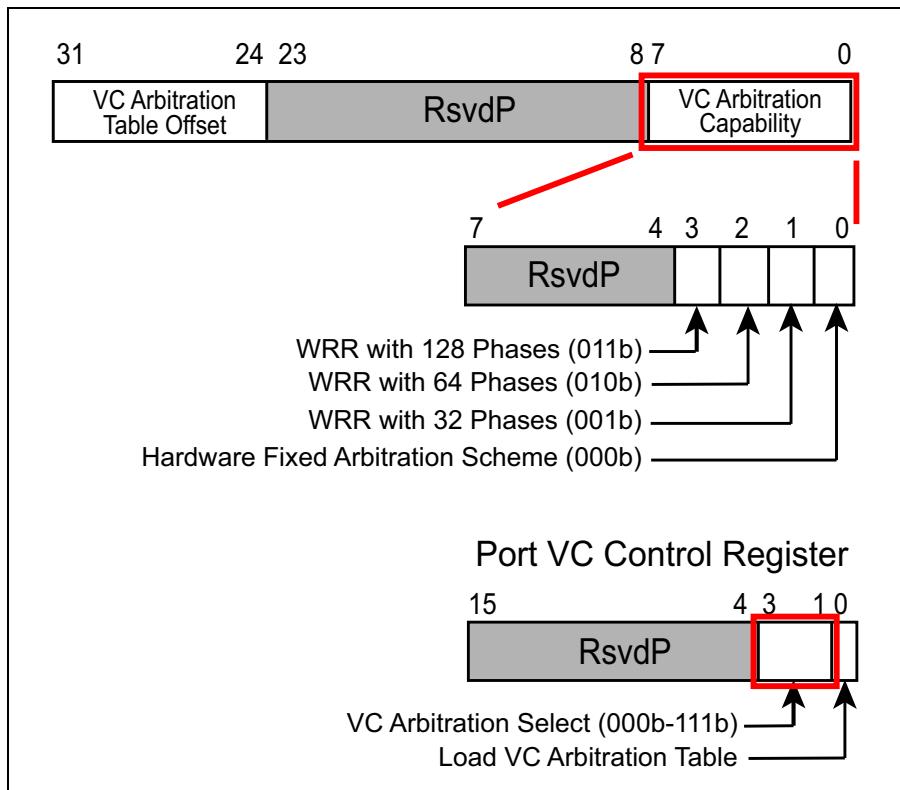


# PCI Express Technology

As depicted in Figure 7-10 on page 257, the high-priority VCs continue to use strict priority arbitration, while the low-priority arbitration group uses one of the other arbitration methods supported by the device. VC Capability Register 2 reports which alternate methods are supported for this group, as shown in Figure 7-9, and the VC Control Register permits selection of the method to be used. The low-priority arbitration schemes include:

- Hardware Based Fixed Arbitration
- Weighted Round Robin Arbitration (WRR)

Figure 7-9: VC Arbitration Capabilities



*Figure 7-10: VC Arbitration Priorities*

VC Resources	VC IDs	Split Priority
8th VC	VC7	Highest
7th VC	VC6	High-Priority (Strict Priority Scheme)
6th VC	VC5	
5th VC	VC4	Low-Priority VC ID = 4
4th VC	VC3	
3rd VC	VC2	Low-Priority (Alternate Priority Scheme) (Selected by Software)
2nd VC	VC1	
1st VC	VC0	Lowest

## Hardware Fixed Arbitration Scheme

This selection defines a hardware-based method and requires no additional software setup. This method can be anything the hardware designer chooses to build in, and could be based on anticipated loading or bandwidth needs for the device. A simple example might be an ordinary round robin sequence, in which each VC gets an equal turn at sending packets during the rotation.

## Weighted Round Robin Arbitration Scheme

This is a scheme in which some VCs can be weighted more (given higher priority) than others by giving them more entries in the sequence than others. The spec defines three WRR options, each with a different number of entries (called phases). The table size is selected by writing the corresponding value in to the *VC Arbitration Select* field of the Port VC Control Register (see Figure 7-9 on page 256). Each entry in the table represents one phase that software loads with a low priority VC number. The VC arbiter will repeatedly scan all table entries in a sequential fashion and send packets from the VC specified in the table entries. Once a packet has been sent, the

arbiter immediately proceeds to the next phase. Figure 7-11 on page 258 shows an example of a WRR arbitration table with 64 entries.

Figure 7-11: WRR VC Arbitration Table

Phase	VC ID
0	VC 4
1	VC 3
2	VC 2
3	VC 1
4	VC 4
5	VC 3
6	VC 0
7	VC 4
8	VC 3
9	VC 2
10	VC 1
11	VC 4
62	VC 3
63	VC 0

↓

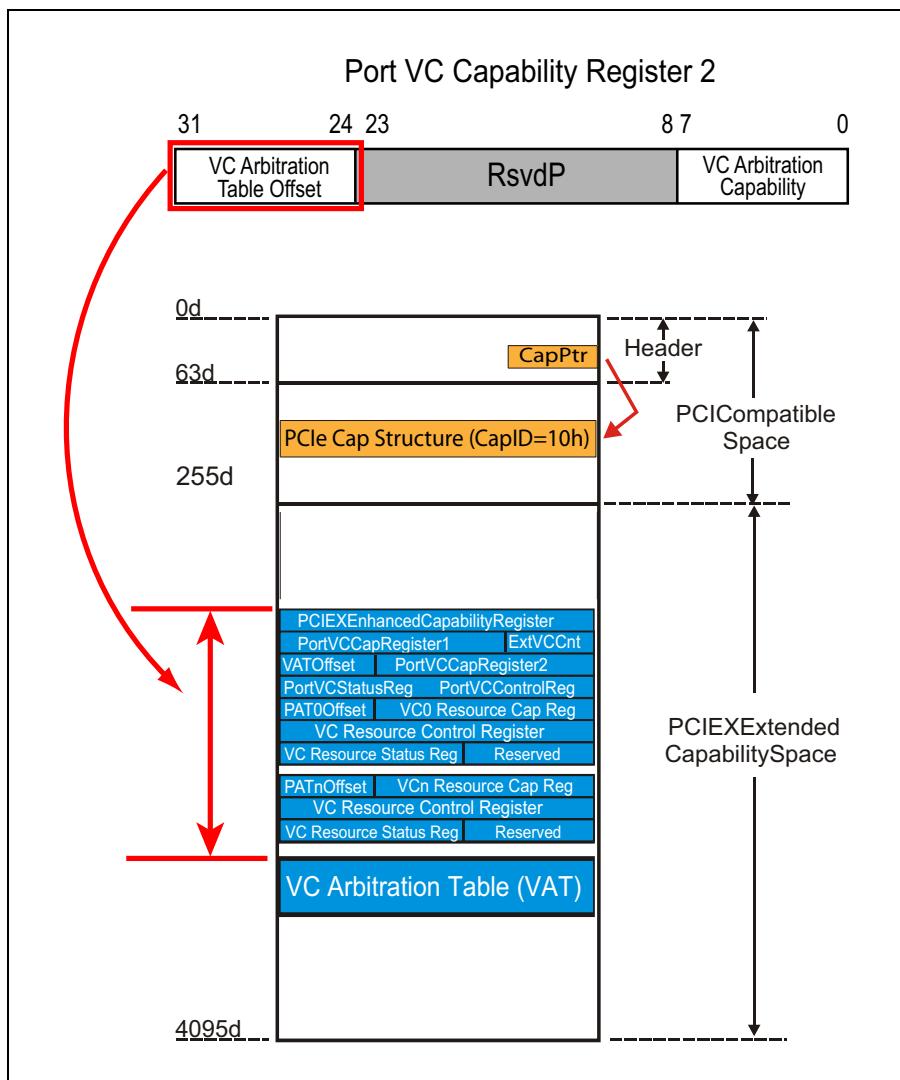
Arbitration Logic Scans Table Entries

## Setting up the Virtual Channel Arbitration Table

The location of the VC Arbitration Table (VAT) in configuration space is given as an offset from the base address of the VC Capability Structure, as shown in Figure 7-12 on page 259.

As shown in Figure 7-13 on page 260, each entry in the VAT is a 4-bit field that identifies the VC number of the buffer that is scheduled to deliver data during that phase. The table length is selected by the arbitration option shown in Figure 7-9 on page 256.

*Figure 7-12: VC Arbitration Table Offset and Load VC Arbitration Table Fields*

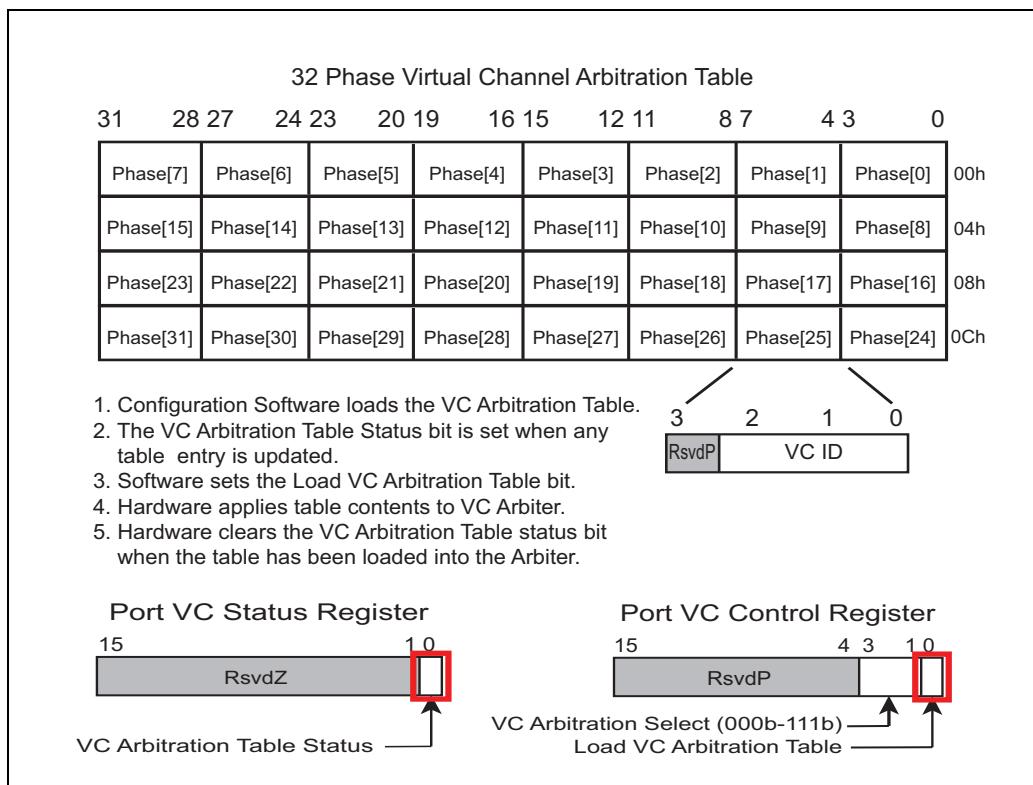


The table is loaded by configuration software to achieve the desired priority order for the virtual channels. Hardware sets the *VC Arbitration Table Status* bit whenever any changes are made to the table, giving software a way to verify whether changes have been made but not yet applied to the hardware. Once the table is loaded, software sets the *Load VC Arbitration Table* bit

# PCI Express Technology

in the Port VC Control register. That causes hardware to load, or apply, the new values to the VC Arbiter. Hardware clears the *VC Arbitration Table Status* bit when table loading is complete, signaling to software that loading has finished. This method is probably motivated by the desire to change the table contents during run time without disruption. The problem is that configuration writes are only able to update a dword at a time and are relatively slow transactions, which means it could take a long time to finish making changes, during which the table is only partially updated. That, in turn, could result in unexpected behavior by the device as it continues to operate during this time. To avoid that, this mechanism allows software to complete all the changes to the table and then apply them all at once to the hardware arbiter.

Figure 7-13: Loading the VC Arbitration Table Entries



---

## Port Arbitration

---

### General

Switch ports and root ports will often receive incoming packets that need to be routed to another port. Since packets arriving from multiple ports can all target the same VC in the same outgoing port, arbitration is needed to decide which incoming port's packet gets next access to that VC. Like VC arbitration, port arbitration has several optional schemes available for selection by configuration software. The combination of TCs, VCs, and arbitration support a range of service levels that fall into two broad categories:

**1. Asynchronous** — Packets get “best effort” service and may receive no preference at all. Many devices and applications, like mass storage devices, have no stringent requirements for bandwidth or latency and don't need special timing mechanisms. On the other hand, packets generated by more demanding applications can still be prioritized without much trouble by establishing a hierarchy of traffic classes for different packets. Differentiated service is still considered to be asynchronous until the level of service requires guarantees. Naturally, asynchronous service is always available and doesn't need any special software or hardware options.

**2. Isochronous** — When latency and bandwidth guarantees are needed, we move into the isochronous category. This would be useful when a synchronous connection would normally be required between two devices. For example, a CD-ROM sourcing data from a music CD uses a synchronous connection when a headset is plugged directly into the drive. However, when the audio must be routed across a general-purpose bus like PCIe to get to external speakers, the connection cannot be synchronous because other traffic may also need to use the same data stream. To achieve an equivalent result, isochronous service must guarantee proper delivery of the time-sensitive audio data without preventing other traffic from using the Link during the same time. Not surprisingly, specialized software and hardware are needed to support this.

The concept of port arbitration is pictured in Figure 7-14 on page 262. Note that port arbitration exists in several places in a system:

- Egress ports of switches
- Root Complex ports when peer-to-peer transactions are supported
- Root Complex egress ports that lead to targets such as main memory

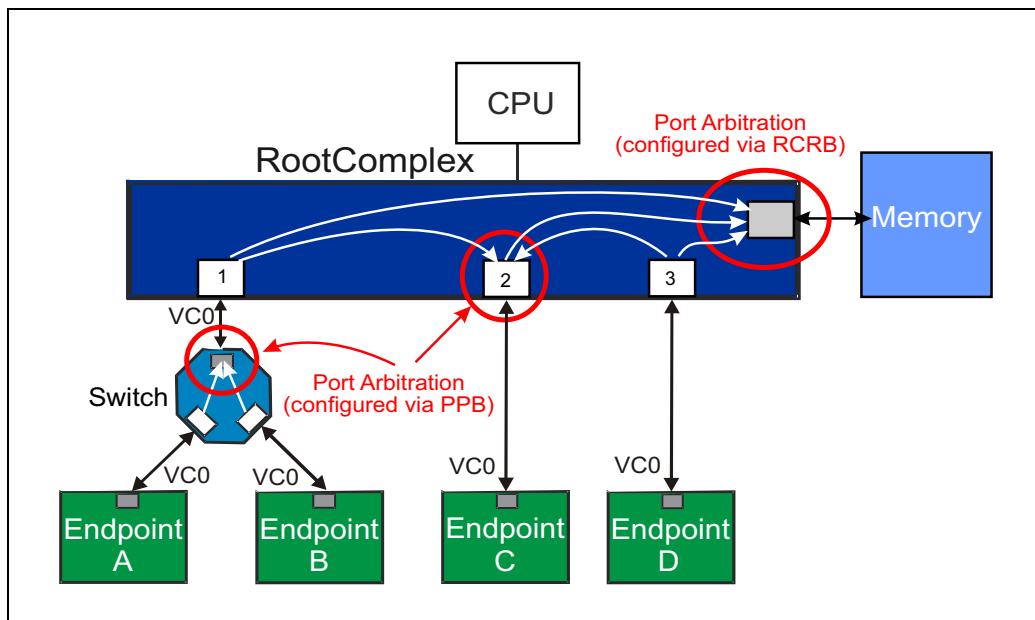
# PCI Express Technology

Port arbitration will usually need software configuration for each virtual channel supported by a switch or root egress port. In the example below, root port 2 supports peer-to-peer transfers from root ports 1 and 2 and therefore needs port arbitration. It should be noted, though, that peer-to-peer support between root ports is optional, so it may be that not every root egress port would need port arbitration.

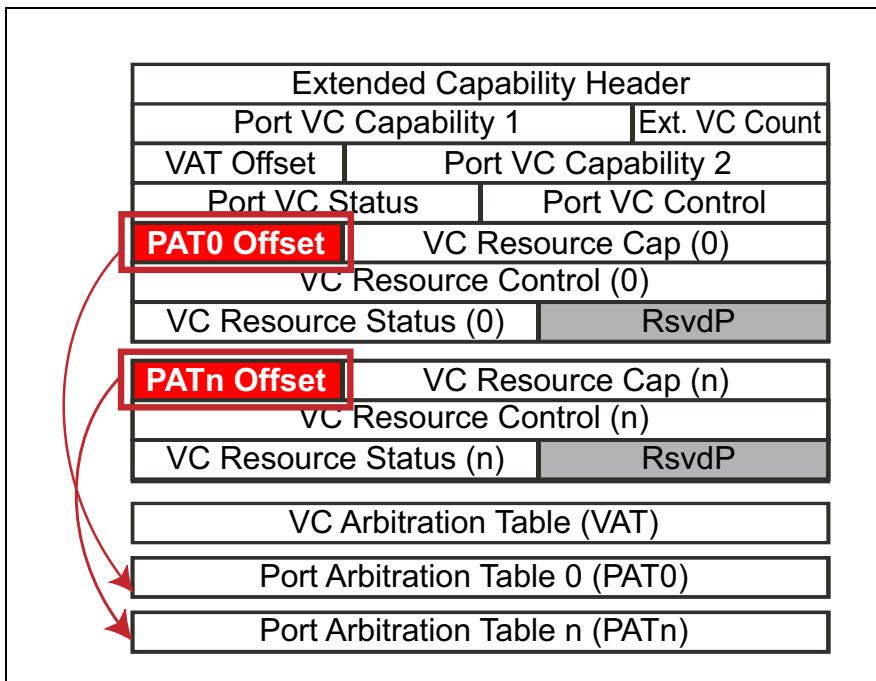
The connection to system memory is an interesting path. There will likely be packets from multiple ingress ports trying to access this port at the same time, so it needs to support port arbitration. However, it doesn't use a PCIe port, so it doesn't have the set of PCIe registers to support arbitration that we're describing here. Instead, the root will need to supply a vendor-specific set of registers called a Root Complex Register Block (RCRB) to provide the same functionality.

Because port arbitration is managed independently for each VC of the egress port, a separate table is required for each VC that supports programmable port arbitration, as shown in Figure 7-15 on page 263. Port arbitration tables are supported only by switches and root ports and are not allowed in endpoints.

Figure 7-14: Port Arbitration Concept



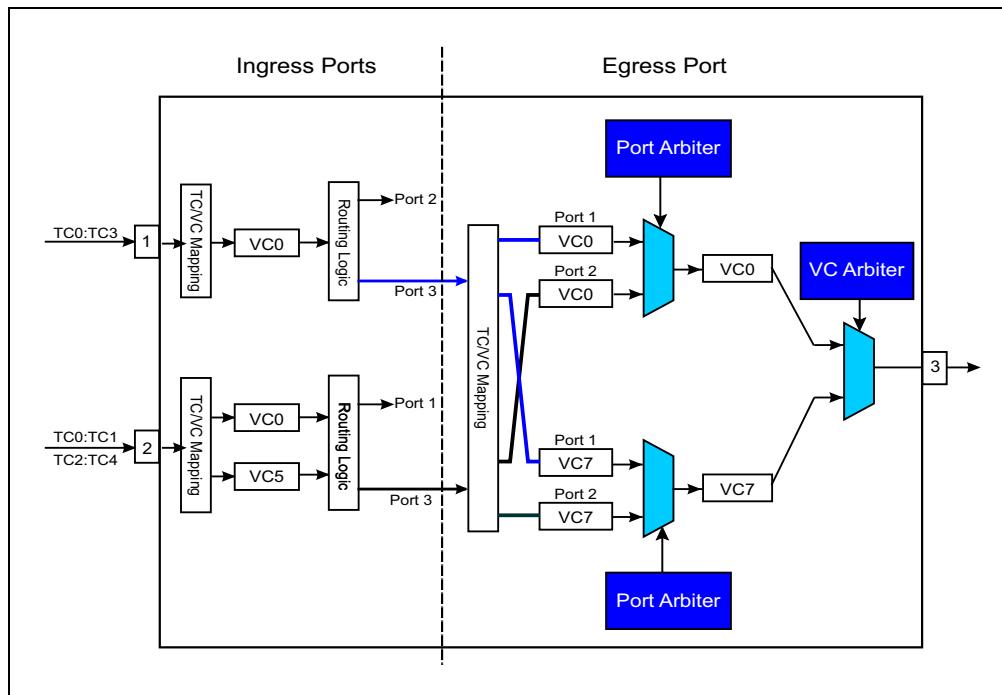
*Figure 7-15: Port Arbitration Tables for Each VC*



Although it isn't stated in the spec, the process of arbitrating between different packet streams also implies the use of additional buffers to accumulate traffic from each port in the egress port as illustrated in Figure 7-16 on page 264. This example illustrates two ingress ports (1 and 2) whose transactions are routed to an egress port (3). The actions taken by the switch include the following:

1. Packets arriving at the ingress ports are directed to the appropriate flow control buffers (VC) based on the TC/VC mapping.
2. Packets are forwarded from the flow control buffers to the routing logic, which determines and routes them to the proper egress port.
3. Packets routed to the egress port (3) use TC/VC mapping to determine into which VC buffer they should be placed.
4. A set of buffers is associated with each of the ingress ports, allowing the ingress port number to be tracked until port arbitration can be done.
5. Port arbitration logic determines the order in which transactions are sent from each group of ingress buffers.

Figure 7-16: Port Arbitration Buffering



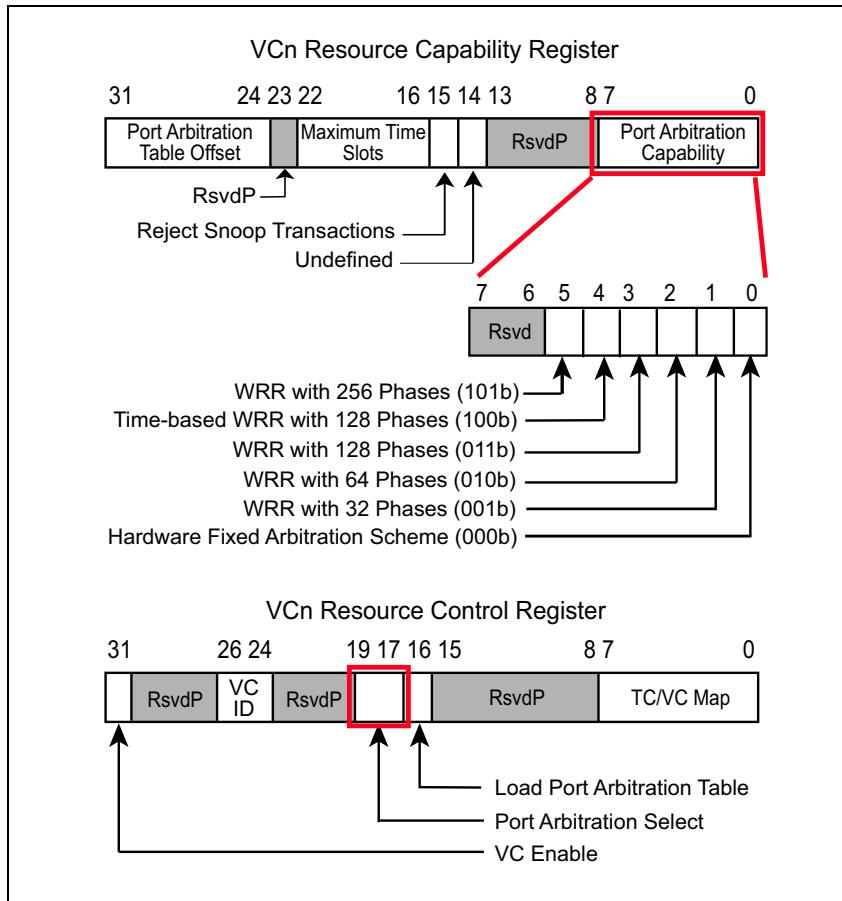
---

## Port Arbitration Mechanisms

The actual port arbitration mechanisms defined are similar to the models used for VC arbitration. Configuration software determines the capability for a port by reading the registers shown in Figure 7-17 on page 265 and selects the port arbitration scheme to use for each VC.

# Chapter 7: Quality of Service

Figure 7-17: Software Selects Port Arbitration Scheme



## Hardware-Fixed Arbitration

This mechanism doesn't require software setup. Once selected, it's managed solely by hardware. The actual arbitration scheme is chosen by the hardware designer, possibly based on the expected demands for the device. This may simply ensure fairness or it may optimize some aspect of the design, but it doesn't support differentiated or isochronous services.

## Weighted Round Robin Arbitration

Just like the weighted round robin mechanism in VC arbitration, software can set up the port arbitration table so that some ports receive more oppor-

tunities than others. This approach assigns different weights to traffic coming from different ports.

As the table is scanned, each phase specifies the port number from which the next packet is received. Once the packet is delivered, the arbitration logic immediately proceeds to the next phase. If no transaction is pending transmission for the selected port, the arbiter advances immediately to the next phase. There is no time value associated with these entries.

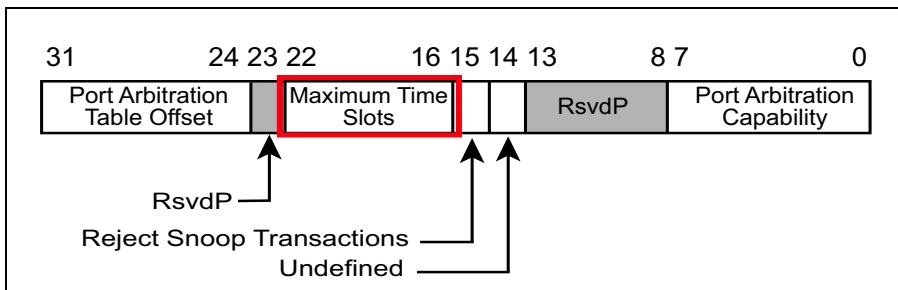
Four table lengths are given for WRR port arbitration, determined by the number of phases used by the table. Presumably, a larger number of entries in the table allows for more interesting ratios of arbitration selection. On the other hand, a smaller number of entries would use less storage and cost less.

## Time-Based, Weighted Round Robin Arbitration (TBWRR)

This mechanism is required for isochronous support. As the name implies, time-based weighted round robin adds the element of time to each arbitration phase. Just as in WRR the port arbiter delivers one transaction from the ingress port VC buffer indicated by the Port Number of the current phase. Now though, rather than immediately advancing to the next phase, the time-based arbiter waits until the current virtual timeslot elapses before advancing. This ensures that transactions are accepted from the ingress port buffer at regular intervals. If the selected port does not have a packet ready to send then nothing will be sent until the next timeslot. Note that the timeslot does not govern the duration of the transfer, but rather the interval between transfers. The maximum duration of a transaction is the time it takes to complete the round robin and return to the original timeslot. The length of the timeslot may change in the future, but currently has the value of 100ns.

Time-based WRR arbitration supports a maximum table length of 128 phases, but the actual number of table entries available for a given VC may be less than that. The value is hardware initialized and reported in the *Maximum Time Slots* field of each virtual channel that supports TBWRR, as shown in Figure 7-18 on page 267.

Figure 7-18: Maximum Time Slots Register



---

## Loading the Port Arbitration Tables

The actual size and format of the Port Arbitration Tables are a function of the number of phases and the number of ingress ports supported by the Switch, RCRB, or Root Port that supports peer-to-peer transfers. The maximum number of ingress ports supported by the Port Arbitration Table is 256 ports. The actual number of bits within each table entry is design dependent and governed by the number of ingress ports whose transactions can be delivered to the egress port. The size of each table entry is reported in the 2-bit *Port Arbitration Table Entry Size* field of Port VC Capability Register 1. The permissible values are:

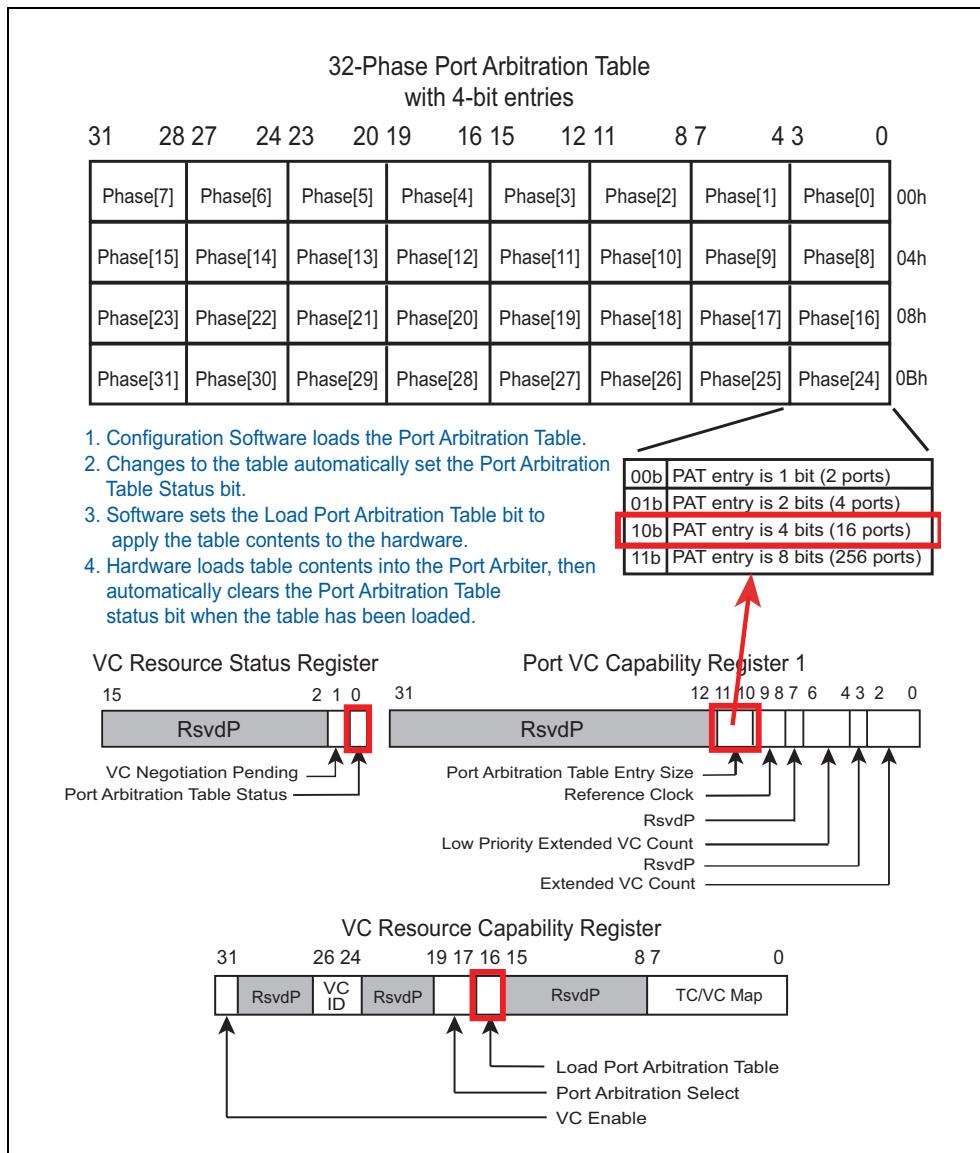
- 00b — 1 bit (selects between 2 ports)
- 01b — 2 bits (4 ports)
- 10b — 4 bits (16 ports)
- 11b — 8 bits (256 ports)

Configuration software loads each table with port numbers to accomplish the desired port priority for each VC supported. As illustrated in Figure 7-19 on page 268, the table format depends on the size of each entry and the number of phases supported by this design.

# PCI Express Technology

---

Figure 7-19: Format of Port Arbitration Tables



## Switch Arbitration Example

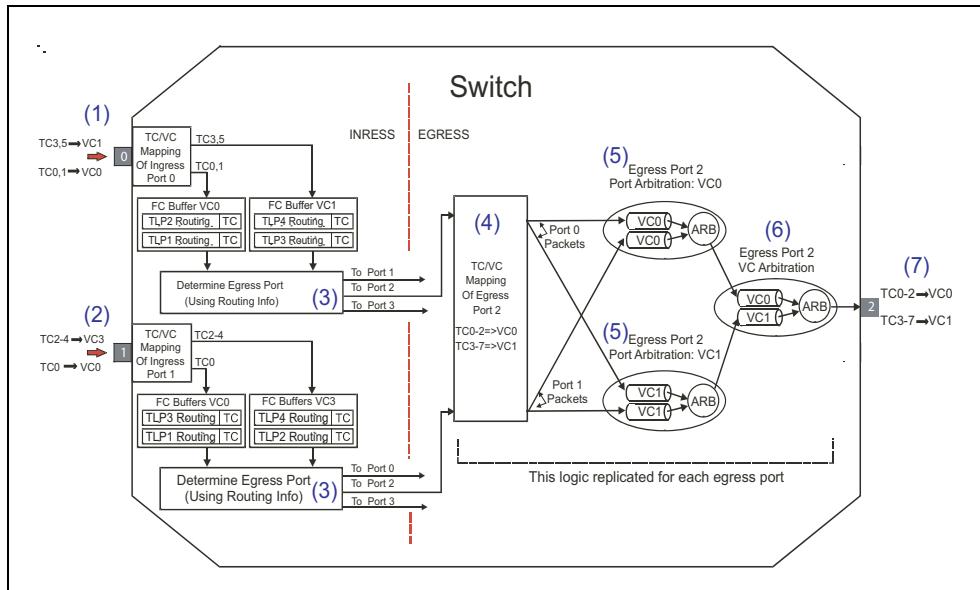
Let's consider an example of a three-port switch to illustrate both Port and VC arbitration. The example presumes that packets arriving on ingress ports 0 and 1 are moving in the upstream direction and port 2 is the egress port facing upstream (toward the Root Complex). Refer to Figure 7-20 on page 270 during the following discussion.

1. Packets arriving at ingress port 0 are placed in a receiver VC based on the TC/VC mapping for port 0. As shown, TLPs with traffic class TC0 or TC1 are sent to the VC0 buffers. TLPs carrying traffic class TC3 or TC5 are sent to the VC1 buffers. No other TCs are permitted on this link. As an aside, if a packet does arrive with a TC that has not been mapped to an existing VC, it will be treated as an error.
2. Packets arriving at ingress port 1 are placed in a VC based on TC/VC mapping, too, but it's not the same for this port. As indicated, TLPs carrying traffic class TC0 are sent to VC0, while TLPs carrying traffic class TC2-TC4 are sent to VC3. No other TCs are permitted on this link.
3. In both ports, the target egress port is determined from routing information in each packet. For example, address routing is used in memory or IO request TLPs.
4. All packets destined for egress port 2 are submitted to the TC/VC mapping logic for that port. As shown, TLPs carrying traffic class TC0-TC2 are placed into buffers for VC0 that are labeled with their ingress port number, while TLPs carrying traffic class TC3-TC7 are managed for VC1.
5. Port Arbitration is applied independently to queued up packets to decide which port's packets will get loaded next into the real VC.
6. Finally, VC arbitration determines the order in which transactions in the VC buffers will be sent across the link.
7. Note that the VC arbiter selects packets for transmission only if sufficient flow control credits exist.

# PCI Express Technology

---

Figure 7-20: Arbitration Examples in a Switch

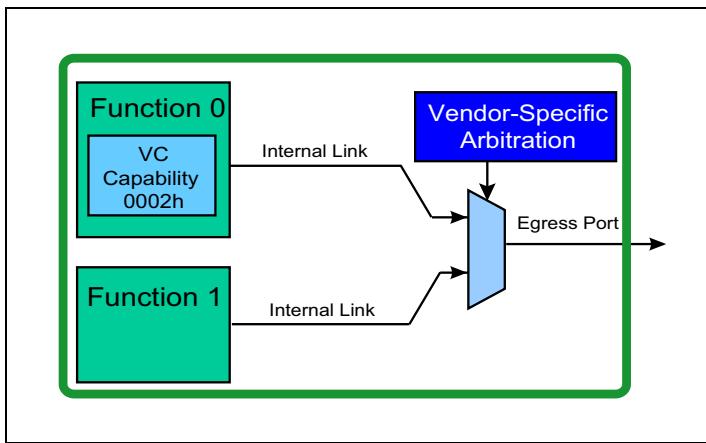


## Arbitration in Multi-Function Endpoints

Another set of registers called Multi-Function Virtual Channel (MFVC) capability is defined for the specific case of endpoints that will implement QoS in a device with multiple functions. Not surprisingly, this case presents the same arbitration issues internally that a switch port must handle.

There are two cases described in the spec for this arbitration. In the first case, shown in Figure 7-21 on page 271, there are two Functions but only Function 0 includes VC Capability registers and the assignments made there are implicitly the same for all functions. For this option, arbitration between the functions will be handled in some vendor-specific manner. That's the simplest approach, but doesn't include a standard structure to define priority between requests from different functions and so it doesn't support QoS.

Figure 7-21: Simple Multi-Function Arbitration



If QoS support is desired, then an MFVC is implemented in VC0 and each function has its own unique set of VC Capability registers. To preserve software backward compatibility, the spec states that the VC Capability ID for a device that *does not* use MFVC must be 0002h, while the VC Capability ID for a device that *does* implement an MFVC structure must be 0009h.

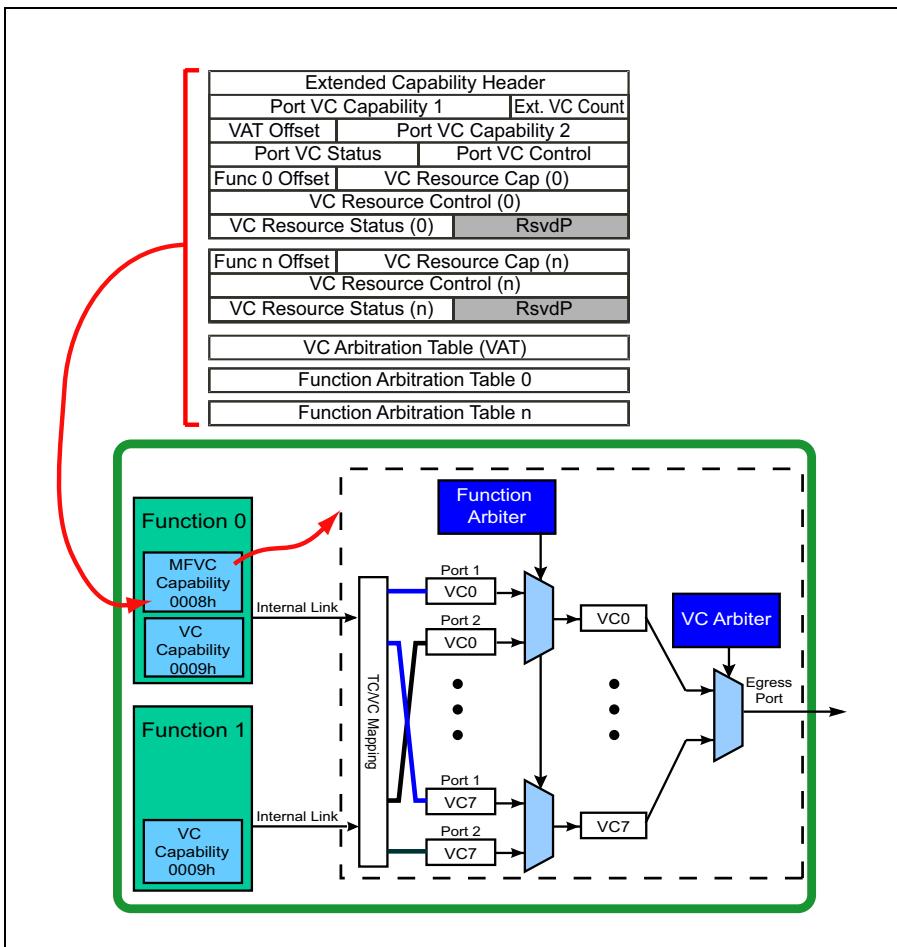
Figure 7-22 on page 272 shows the MFVC register block and a block diagram of an example with two functions in an endpoint whose port supports two VCs. Each function has a Transaction Layer and its own VC Capability registers, but doesn't implement the lower layers. Instead, they connect to the Transaction Layer of the shared port that does have all the layers. Sharing the hardware interface results in lower cost, of course, and the addition of MFVC allows the functions to handle isochronous traffic.

As can be seen in the figure, the MFVC registers reside in Function 0 only and define the VCs and arbitration methods to be used for this interface. The MFVC registers look very much the same as VC capability registers and support VC arbitration and Function arbitration. Since packets from multiple functions can attempt to access the same VC at the same time, Function Arbitration decides the priorities among them. That should look familiar by now because it's the same concept as port arbitration and even uses the same arbitration options, including TBWRR. VC arbitration options are also the same as they are in the single-function VC registers.

# PCI Express Technology

---

Figure 7-22: QoS Support in Multi-Function Arbitration



---

## Isochronous Support

As mentioned earlier, not every machine or application needs isochronous support, but there are some that can't get by without it. Since PCIe was designed to support it from the beginning, let's consider what would need to be in place to make this work.

## Timing is Everything

Consider the example shown in Figure 7-23 on page 274, where a synchronous connection would be desirable but isn't possible. Instead, we emulate a synchronous path with isochronous mechanisms. In this example, isochrony defines the amount of data that will be delivered within each Service Interval to achieve the required service. The following sequence describes the operation:

1. The synchronous source (video camera and PCI Express interface) accumulates data in Buffer A during the first of the equal service intervals (SI 1).
2. The camera delivers all of the accumulated data across the general-purpose bus during the next service interval (SI 2) while it accumulates the next block of data in Buffer B.

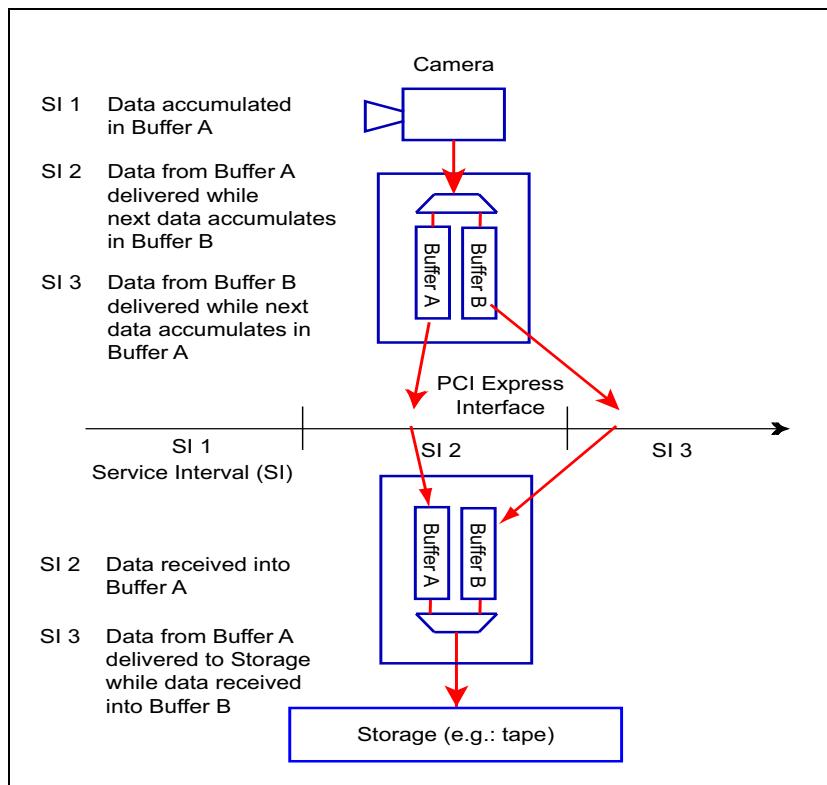
Clearly, the system must be able to guarantee that the entire contents of buffer A can be delivered during the service interval, regardless of whether other traffic is in flight on the Link. This is handled by assigning a high priority to the time-sensitive packets and programming arbitration schemes so they'll be handled first any time there is competition with other traffic. Also note that, as long as all the data is delivered within the time window, it doesn't matter exactly when it arrives. It might be spread out across the interval or bunched up in one place inside it. As long as it's all delivered with the Service Interval the guarantees can still be met.

3. During SI 2, the tape deck receives and buffers the incoming data, which can then be delivered to storage for recording during SI 3. The camera unloads Buffer B onto the Link during SI 3 while accumulating new data into Buffer A, and the cycle repeats.

# PCI Express Technology

---

Figure 7-23: Example Application of Isochronous Transaction



## How Timing is Defined

Isochronous timing is defined in PCIe by the time slot used in the Time-Based Weighted Round Robin port arbitration scheme. At present, the time for each slot is 100ns, and represents one entry of the 128 entries in the TBWRR table. Once set up, the arbiter will repeatedly cycle through this table once every 12.8μs, which represents the overall Service Interval.

Making an isochronous path work as intended requires a few considerations. First, the data packets must be delivered with predictable timing at regular intervals. Second, the maximum amount of isochronous data to be delivered must be known ahead of time and packets must not be allowed to exceed that limit. Third, the Link bandwidth must be sufficient to support the amount of data that needs to be delivered in a given time slot.

Consider the following example. A single-Lane Link running at 2.5 Gbps delivers one symbol every 4ns. That allows it to send 25 symbols within a 100ns time slot, but is that enough to be useful? In many cases it's not, because a TLP may need 28 bytes of overhead for the combination of header, sequence number, LCRC, and so forth. That would mean there isn't even time to finish sending the overhead, much less any data payload in 100ns. If we needed to send 128 bytes of data, then the bandwidth requirement would be  $128 + \text{overhead} = 156$  bytes. One option for solving this problem would be to increase the Link width to 8 Lanes, allowing eight times as many bytes to be sent at once. That change would deliver 200 bytes in 100ns and allow a single time slot to deliver all the isochronous data. Another solution would be to use a single Lane but give the port more time slots, since 8 time slots at the lower Link width would deliver the same amount of data. The choice of solution depends on cost and performance constraints, but the system designer must know the timing and bandwidth requirements of the isochronous path to be able to set it up correctly.

## How Timing is Enforced

When timing is an integral part of the proper operation of a design, as in the previous example, it is enforced by the combination of things we've discussed so far. First, high-priority TCs must be selected in software and VCs set up in hardware with the mappings between them defined so that only the correct packets will be placed into the high-priority VCs. Then the desired timing is a matter of programming the arbitration schemes to accommodate the needed bandwidth in the specified time. For example, the choice for VC arbitration would probably be the Strict Priority option, since it's the only choice that can ensure that a high-priority packet won't be delayed by other packets. For Port arbitration the choice must be TBWRR to enforce timing.

---

## Software Support

Supporting isochronous service requires some coordination between the software elements in the system. In a PC system, device drivers will report isochronous requirements and capabilities to the OS, which will then evaluate the overall system demands and allocate resources appropriately. Embedded systems will be different, because the all the pieces are known at the outset and software can be simpler. In the following discussion we'll describe the PC case since an embedded system should simply be a simpler subset of that.

# PCI Express Technology

---

## Device Drivers

A device driver must be able to report its timing requirements to the software that oversees isochronous operation and obtain permission before trying to use isochronous packets. It's important to note that driver-level software should not directly change hardware assignments or arbitration policies on its own, even though it could, because the result would be chaos. If multiple drivers were each independently trying to do this, the last one to make changes would overwrite any previous assignments. To avoid that, an OS-level program called an Isochronous Broker receives the timing requests from the system devices and assigns system resources in a coordinated way that accommodates them all.

## Isochronous Broker

This program manages the end-to-end flow of isochronous packets. It receives the isochronous timing requests from device drivers and allocates system resources in a way that accommodates the requests through the target path. In the spec this is referred to as establishing an isochronous contract between the requester/completer pair and the PCIe fabric. Doing so requires verifying that the intended path can indeed support isochronous traffic, and then programming the appropriate arbitration schemes to ensure it works within the specified timing requirements.

---

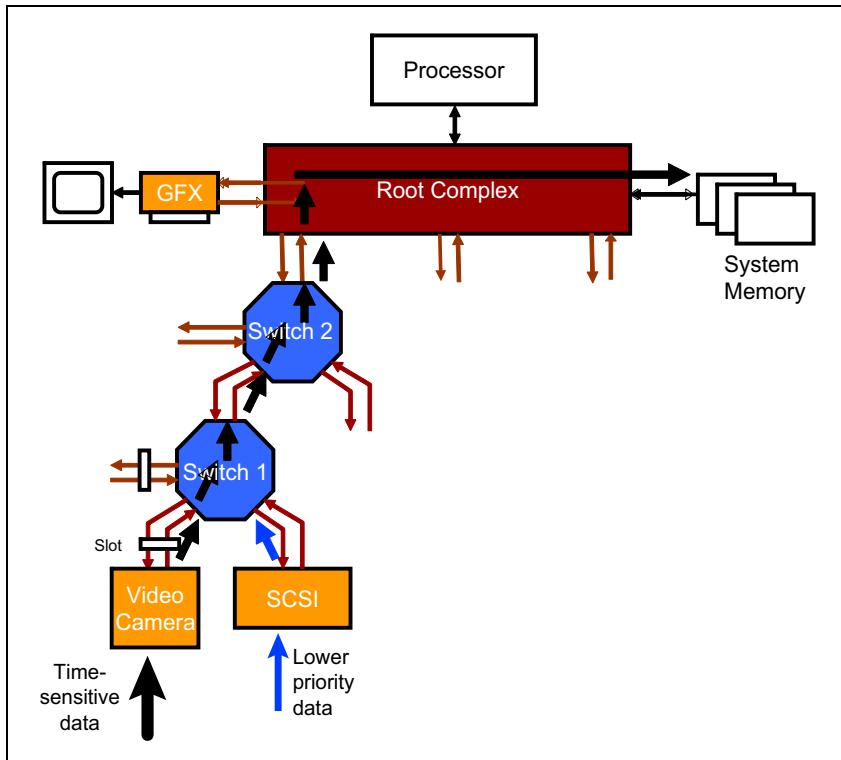
## Bringing it all together

By now it should be reasonably clear what needs to be done to support isochronous traffic flow in a system, but let's look at one last example to bring all the pieces together. If we expand on the previous video capture example to show a more complex system, like the one in Figure 7-24 on page 277, we'll be able to discuss all the parts that must be in place if the video camera is going to be able to deliver captured data into system memory. This would be a difficult environment for isochronous service because there are so many devices that can compete for bandwidth in the path, but that also makes it useful to illustrate the various things that must be considered.

## Endpoints

Starting at the bottom, what will be needed in the PCIe interface for the video endpoint device itself? In hardware, more than one VC will be required if we're going to differentiate packets. Let's assume a single-function device for simplicity. The device driver would need to report the device capabilities and isochronous timing requirements to the OS-level Isochronous broker, which would evaluate the system and then report back whether an isochronous contract was possible and which TCs the software should use.

Figure 7-24: Example Isochronous System



The driver would then program VC numbers and map the appropriate TCs to each VC. It would also most likely program the VC arbitration to be Strict Priority for the high-priority channels. The one caveat here is that the arbitration must still be “fair”, meaning the low-priority channels won’t get starved for access. That means the high-priority VCs can’t have traffic pending constantly but instead must spread out packet injection over time.

One other observation regarding Link operation is necessary before we finish our discussion of endpoints, and that is regarding Flow Control. The receive buffers of devices in the isochronous path must be large enough to handle the expected packet flow without causing any back pressure as long as packets are injected uniformly according to the Isochronous Contract. In addition, Flow Control Updates must be returned quickly enough to avoid stalls.

## Switches

Next, consider what would need to be present in each of the switches that reside between the endpoint and the Root Complex. Switches don't commonly have device drivers, so it would fall to OS-level software like the Isochronous Broker to read their configuration information and determine what service they support. First, all the ports in the isochronous path must support more than one VC, and the TC/VC mapping must match on both ends of each Link. Remember that once the packet gets into the Transaction Layer of the Switch port, only the TC remains with the packet, and the VC assignment for that TC is specific to each port. The TC/VC mapping of the downstream port of Switch 1 must match the mapping of the endpoint, but the other switch port mappings may be different to match the other end of their Links.

**Arbitration Issues.** The choices for arbitration are straight-forward. In our example, the isochronous path is shown as carrying traffic in only one direction for simplicity. It is possible to have isochronous traffic flowing in both directions in the case of a memory read, for example, but our example was chosen to resemble the video streaming case.

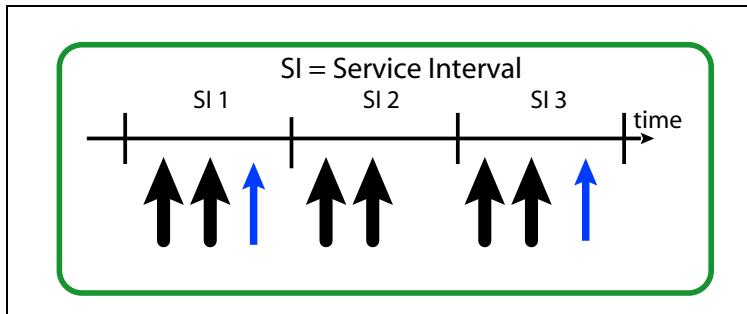
VC arbitration for the isochronous egress port will most likely need to use the Strict Priority scheme for the same reasons the endpoint does. Port arbitration will need to use the Time-Based WRR scheme, and that means software must understand the proper access ratios and program the Port Arbitration Tables to implement them. This might not be as simple as it sounds if multiple switches are in the path because even though they'll all use the same TBWRR arbitration scheme, it's not clear how the service intervals for each of them would be coordinated. If the SIs are not aligned, meaning timing guarantees could be more difficult depending on the how busy the Links are. Coordinating the service intervals wasn't considered in the spec, though, so it would again involve a non-standard method. Clearly, this problem would be much simpler if we didn't have multiple switches in an isochronous path.

**Timing Issues.** Figure 7-25 on page 279 shows the timing of packets being delivered by the two endpoints for our example. Packets from the video device, with a known size and delivered in regular and predictable intervals, are shown as the heavier arrows. The smaller, lighter arrows represent packets from the SCSI drive that are lower priority and whose timing is not predictable. In the endpoint, the packets simply need to have the proper TC assigned to them, but a switch needs to ensure that the proper timing policy is enforced. This is done by using TBWRR, which specifies which port will have access at a given time and for how long. Knowing the size and fre-

## Chapter 7: Quality of Service

quency of the isochronous packets allows software to properly arrange the timing, but what kind of timing is needed?

Figure 7-25: Injection of Isochronous Packets



First, let's review the parameters involved by considering a simple example. Recall that PCIe bases a time slot on the reference clock period is given by the Port Capability Register 1 field called Reference Clock. At present the only option for that field is 100ns, and the TBWRR table has no options besides 128 entries. The length of the Service Interval is the multiple of those, making it 12.8μs. The bandwidth for a given device can be expressed by the equation below, where Y is the data to be delivered in one time slot (the spec states that the Max Payload Size programmed during configuration must be used for this bandwidth calculation), M is the number of time slots, and T is the overall Service Interval. If we choose 128 bytes as the payload, and we know that SI is 12.8μs, then the BW = 10 MB/s for each time slot allocated.

$$BW = \frac{Y \times M}{T}$$

Now let's consider a more realistic example. Let's say that our Links are running at the Gen2 speed, that the video device needs to have a guaranteed bandwidth of 100MB/s, and that it will send 512 byte packets. Filling in the equation shows M = 2.5 instances of 512 bytes are needed. But how much data can actually be

$$100 \times 10^6 = \frac{512 \times M}{12.8 \times 10^{-6}}, M = \frac{100 \times 10^6 \times 12.8 \times 10^{-6}}{512} = 2.5$$

sent in one time slot? The answer depends on speed and Link width, or course. At 5.0 Gb/s it takes 2ns to send each 10-bit symbol, so 50 symbols can be delivered per Lane in 100ns. If the packet size is 512 bytes of data plus another 28 or so for the header, then 11 time slots would be needed to deliver 550 symbols for one packet using a x1 Link. It is possible to give one port several contiguous

# PCI Express Technology

---

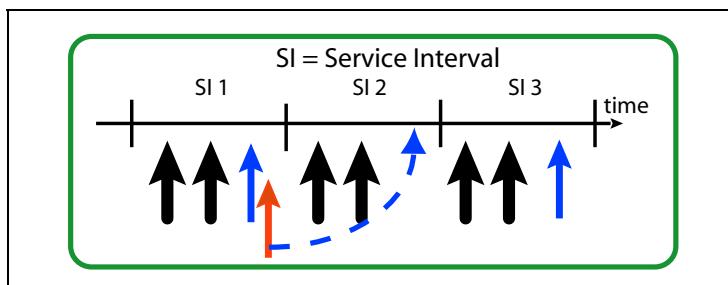
slots if needed, so that's one solution. Since the packet size that will be sent is always the same, we can't really program 2.5 instances of it, so we'd have to use 3 instead. From our equation, 3 instances of 512 bytes each results in an actual bandwidth of 120MB/s. That's higher than we need, but it solves the problem. The number of time slots used would then be  $11 \times 3 = 33$ , leaving 95 for other use in the Service Interval. Each group of 11 time slots would need to be contiguous but the groups could be spaced out over the service interval.

Another solution would be increase the Link width. Although the hardware would cost more, using 11 Lanes would allow delivery of all the data in one time slot. The CEM spec doesn't currently support a x11 option, but a x12 option is available and would work for our example. Using a wide Link like that means software would only need to program one time slot for each packet, and just three over the whole service interval to support isochronous traffic for this device. Unlike the x1 case, now we wouldn't need contiguous time slots. Instead, they could be spaced over the service interval in some optimal fashion.

**Bandwidth Allocation Problems.** The TBWRR table must be programmed to guarantee sufficient timely bandwidth for isochronous traffic, and that other traffic won't be allowed to interfere. In Figure 7-25 on page 279, the SCSI controller is shown as sending one packet in SI 1 and another in SI 3. If the timing was such that one packet from that endpoint per SI was allowed then this works fine.

Now let's say the SCSI controller attempts to inject more packets than it has permission to do in SI 1, illustrated in Figure 7-26 on page 280. This is the first of two bandwidth allocation problems mentioned in the spec and is called "oversubscription." This could interfere with isochronous traffic flow, but programming the TBWRR table readily avoids that problem because the arbitration only allows a packet from that port at specific times. If more packets from that port are queued up, they simply have to wait until the next available time, which might be in SI 2, as shown in this example. Eventually, this can result in flow control back-pressure at the sending agent

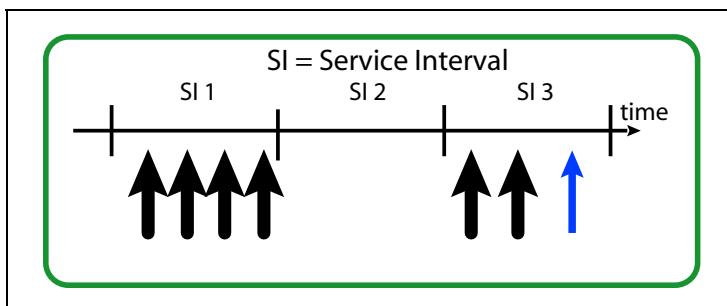
Figure 7-26: Over-Subscribing the Bandwidth



## Chapter 7: Quality of Service

The second timing problem is called “congestion” and happens when too many isochronous requests are sent within a given time window, as shown in Figure 7-27 on page 281. This is a similar problem but now there is no simple solution. Unlike the previous case, postponing high-priority packets until another time slot is not an option, so the system must make an effort to handle them all. The result is that some requests may experience excessive service latencies. To correct this, software would need to change the distribution of packets so that they can be supported by the available hardware bandwidth.

Figure 7-27: Bandwidth Congestion



**Latency Issues**. Managing latency for packet delivery is an important part of isochrony, and involves the combination of the fabric latency and the Completer latency. Fabric latency depends on all the characteristics of the Link between the various components in the system, especially the Link width and frequency of operation. A simple way to minimize this value is to constrain the complexity of the PCIe topology for isochronous paths. Completer latency depends on the target endpoint internal characteristics, such as memory speed and internal arbitration.

### Root Complex

The RC has the same arbitration and timing requirements as a switch. It receives packets on several downstream ports and forwards them to the target in a way that's consistent with the rules for isochrony described earlier. However, much of how this is done will be vendor specific because the spec doesn't define the RC or how it should be programmed.

**Problem: Snooping.** One interesting thing affecting timing and latency in the root that we haven't yet discussed is the process of snooping. Normally, anytime an access to system memory takes place it will be to a location that the processor considers cacheable, meaning it has permission to store a tem-

porary copy in its local caches. If an external device attempts to access that area of memory, the chipset must first check the processor caches before allowing the access because a cached copy may have been modified. If so, the modified data will need to be written back to memory before it will be available for the device access. Although it's necessary to ensure memory coherency, the problem is that snooping takes time. How long it takes is typically bounded but not predictable because it depends on what else the CPUs are doing at that time. Depending on the timing requirements, that kind of uncertainty could ruin an isochronous data flow.

**Snooping Solutions.** One way to avoid snooping is for devices to only access areas of memory that have been designated as uncacheable. Another option is for software to set the "No Snoop" attribute bit in the high-priority packet headers. That forces the chipset to skip the snoop step regardless of the memory type and go directly to memory because software has guaranteed that doing so won't cause a problem. To enforce this as a requirement for the isochronous path, another bit can be initialized by hardware in the root port for the high-priority VC called "Reject Snoop Transactions" (see the VC Resource Capability Register in Figure 7-17 on page 265). The purpose of this is to allow only transactions for that VC that have the No Snoop attribute set. Any incoming packets that don't have it set are discarded to ensure that the timing will never be violated by waiting for a snoop.

## Power Management

It's a simple observation, but if timing is important for a path in PCIe, then power management (PM) mechanisms for devices in that path will need to handled carefully. Configuration software can read the latencies associated with every PM condition and select those cases that the timing budget will permit. The simplest approach, though, would just be to disable all PM options in an isochronous path. Fortunately, this is easily done using existing configuration registers. Devices can be placed into the device state D0 and left there, while the hardware-controlled Link PM mechanism can be disabled (for more on PM, see Chapter 16, entitled "Power Management," on page 703).

## Error Handling

Finally, there is one last issue: what to do when errors occur on the Link. The ACK/NAK protocol, covered in Chapter 7, provides an automatic, hardware-based retry mechanism to correct packets that encounter transmission problems. This otherwise desirable feature presents a problem for isochrony because it takes time to do it. And how long it takes to resolve an error can vary widely depending on things like how the problem was detected.

## Chapter 7: Quality of Service

---

To decide this question we have to know how much time uncertainty the system can tolerate and still deliver isochronous data. If the latency budget is too tight, there simply won't be time for retrying failed packets and the ACK/NAK protocol will have to be disabled. Interestingly, the spec writers evidently didn't consider that possibility because no configuration bits are included for disabling it or deciding how to handle packets that would have been retried but now won't be. Therefore disabling this will require non-standard mechanisms like vendor-specific registers.

If there *isn't* enough time available for retries, the target agent may simply choose to discard any bad packets. Another option would be to use the bad packets as they are, errors and all. For some applications using isochronous support that isn't as counter-intuitive as it sounds. An error in video streaming, for example, might cause an occasional glitch on the display, but that could be considered an acceptable risk.

If there *is* enough time in the Service Interval to allow retries, a limit could be placed on the possible latency they might add by adding a timer to track the time until the end of the Service Interval and use that to decide whether a retry could be attempted. Errors shouldn't happen very often, of course, so this might be sufficient to correct the occasional transmission fault while still maintaining isochronous timing.

## **PCI Express Technology**

---

---

---

# 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 vendor 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.

# PCI Express 3.0 Technology

---

- 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 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.

# Chapter 8: Transaction Ordering

---

*Table 8-1: Simplified Ordering Rules Table*

Row pass Column? (Col 1)	Posted Request (Col 2)	Non-Posted Request		Completion (Col 5)
		Read Request (Col 3)	NPR with Data (Col 4)	
<b>Posted Request</b> (Row A)	a) No b) Y/N	Yes	Yes	a) Y/N b) Yes
Non-Posted Request	Read Request (Row B)	a) No b) Y/N	Y/N	Y/N
	NPR with Data (Row C)	a) No b) Y/N	Y/N	Y/N
<b>Completion</b> (Row D)	a) No b) Y/N	Yes	Yes	a) Y/N b) No

- **A2a, B2a, C2a, D2a** — to enforce the Producer/Consumer model, a subsequent transaction is not allowed to pass a Posted Request.
- **A2, D2b** — If RO is set, then a Read Completion is permitted to pass a previously queued Memory Write or Message Request.
- **A2b, B2b, C2b, D2b** — if the optional IDO is being used, a subsequent transaction is allowed to pass a Posted Request, as long as their Requester IDs are different
- **A3, A4** — A Memory Write or Message Request must be allowed to pass Non-Posted Requests to avoid deadlocks.
- **A5a** — Posted Request is permitted but not required to pass Completions
- **A5b** — Deadlock avoidance case. In a PCIe-to-PCI/PCI-X bridge, for transactions going from PCIe to PCI or PCI-X, a Posted Request must be able to pass a Completion, or a deadlock may occur.
- **B3, B4, B5, C3, C4, C5**, — These cases implement weak ordering without risking any ordering related problems.
- **D3, D4** — Completions must be allowed to pass Read and I/O or Configuration Write Requests (Non-Posted Requests) to avoid deadlocks.
- **D5a** — Completions with different Transaction IDs may pass each other.
- **D5b** — Completions with the same Transaction ID are not allowed to pass each other. This ensures that multiple completions for a single request will remain in ascending address order.

## **Producer/Consumer Model**

This section describes the operation of the Producer/Consumer model and the associated ordering rules required for proper operation. Figure 8-1 on page 291 simply illustrates a sample topology. Subsequent examples of this topology describe the operation of the Producer/Consumer model with proper ordering, followed by an example of the model failing due to improper ordering.

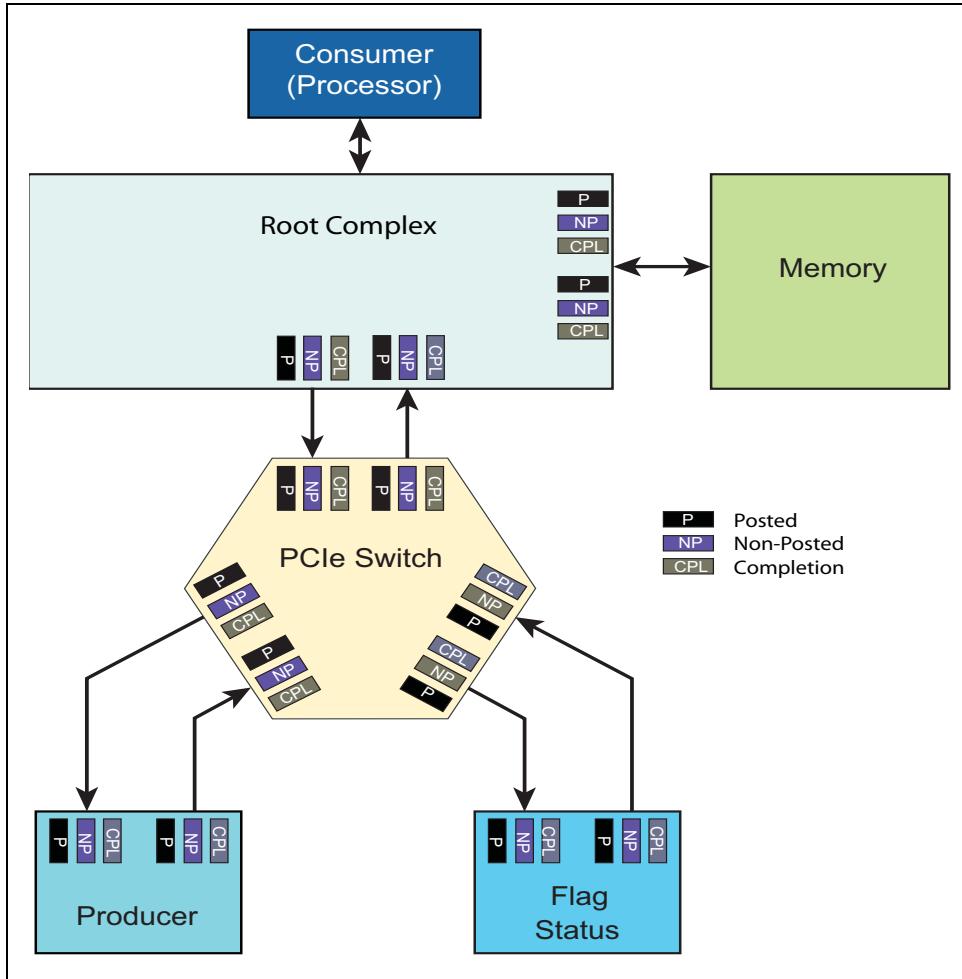
The Producer/Consumer model is the common method for data delivery in PCI and PCIe. The model comprises five elements as depicted in Figure 8-1:

- Producer of data
- Memory data buffer
- Flag semaphore indicating data has been send by the Producer
- Consumer of data
- Status semaphore indicating Consumer has read data

The specification states that the Producer/Consumer model will work regardless of the arrangement of all the elements involved. In this example, the Flag and Status elements reside in the same physical device, but could be located in different devices.

# Chapter 8: Transaction Ordering

Figure 8-1: Example Producer/Consumer Topology



## Producer/Consumer Sequence — No Errors

Refer to Figure 8-2 on page 293 during the following discussion. The example presumes that the Flag and Status element are cleared to start with. These semaphores are included within the same device in this example. The sequence of numbered events in the description below and depicted in Figure 8-2 on page 293 reflect the correct ordering in this Part 1 sequence.

## PCI Express 3.0 Technology

---

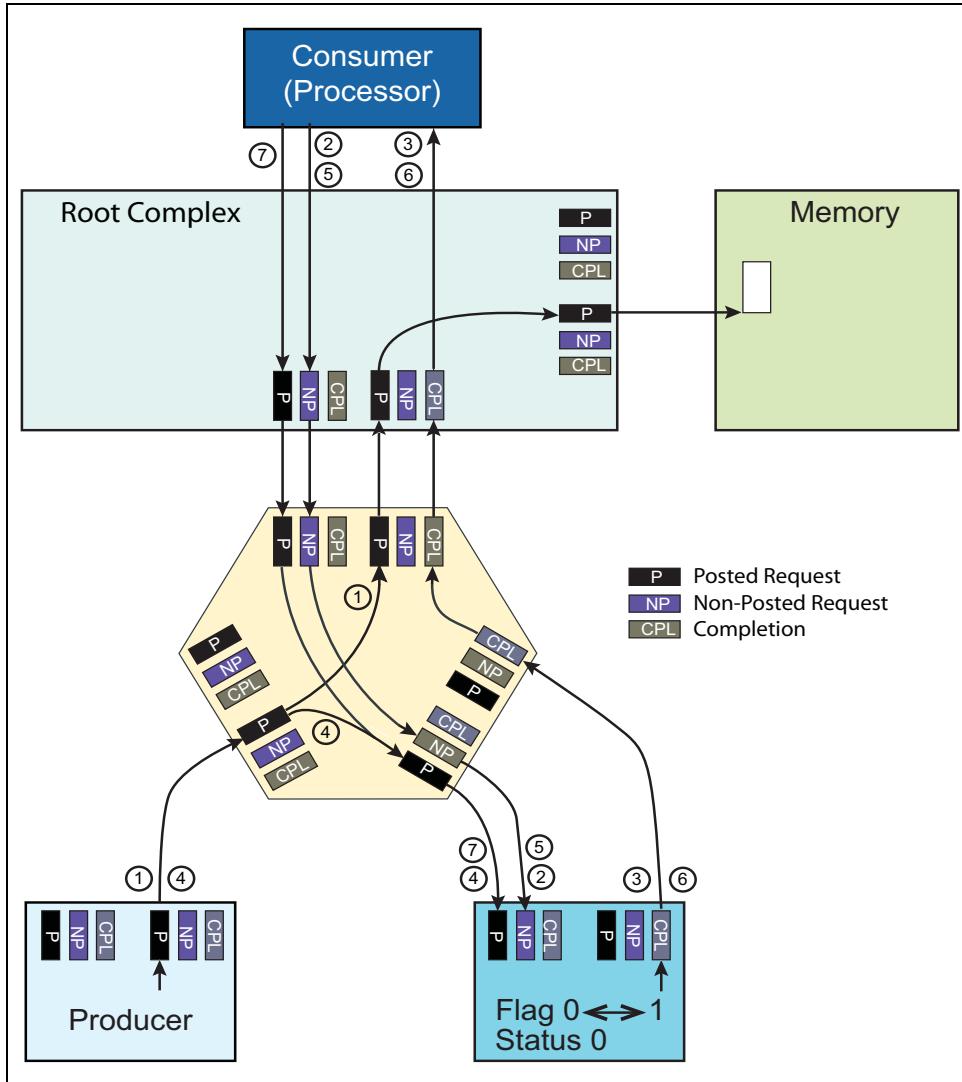
1. In the example, a device called the **Producer** performs one or more Memory Write transactions (Posted Requests) targeting a **Data Buffer** in memory. Some delay can occur as the data flows through Posted buffers.
2. The Consumer periodically checks the Flag by initiating a Memory Read transaction (Non-Posted Request) to determine if data has been delivered by the Producer.
3. The Flag semaphore is read by the device and a Memory Read Completion is returned to the Consumer, indicating that notification of data delivery has not been performed by the Producer (Flag = 0) yet.
4. The Producer sends a Memory Write Transaction (Posted Request) to update the Flag to 1.
5. Once again, the Consumer checks the Flag by performing the same transaction performed in step 2.
6. When Flag semaphore is read this time, the Flag is set to 1, indicating to the Consumer, via the Completion, that all of the data has been delivered by the Producer to memory.
7. Next, the Consumer performs a Memory Write transaction (Posted Request) to clear the Flag semaphore back to zero.

Figure 8-3 on page 294 continues the example in this Part 2 sequence.

8. The Producer, having more data to send, periodically checks the Status semaphore by initiating a Memory Read transaction (Non-Posted Request).
9. The Status semaphore is read by the Producer and a Memory Read Completion is returned to the Producer, indicating that the Consumer has not read the memory buffer contents and updated Status (Status = 0).
10. The Consumer, knowing that the memory buffer has data available, performs one or more Memory Read Requests (Non-Posted Requests) to get the contents from the buffer.
11. Memory contents are read and returned to the Consumer.
12. Upon completing the data transfer, the Consumer initiates a Memory Write Request (Posted Request) to set the Status semaphore to a 1.
13. Once again, the Producer checks the Status semaphore by delivering a Memory Read Request (Non-Posted Request).
14. The device reads the Status and this time it is set to 1. The Completion is returned to the Producer, thereby indicating data can be sent to Memory.
15. The Producer sends a Memory Write to Clear the Status semaphore to 0.
16. The sequence of events starting with step 1. is repeated by the Producer.

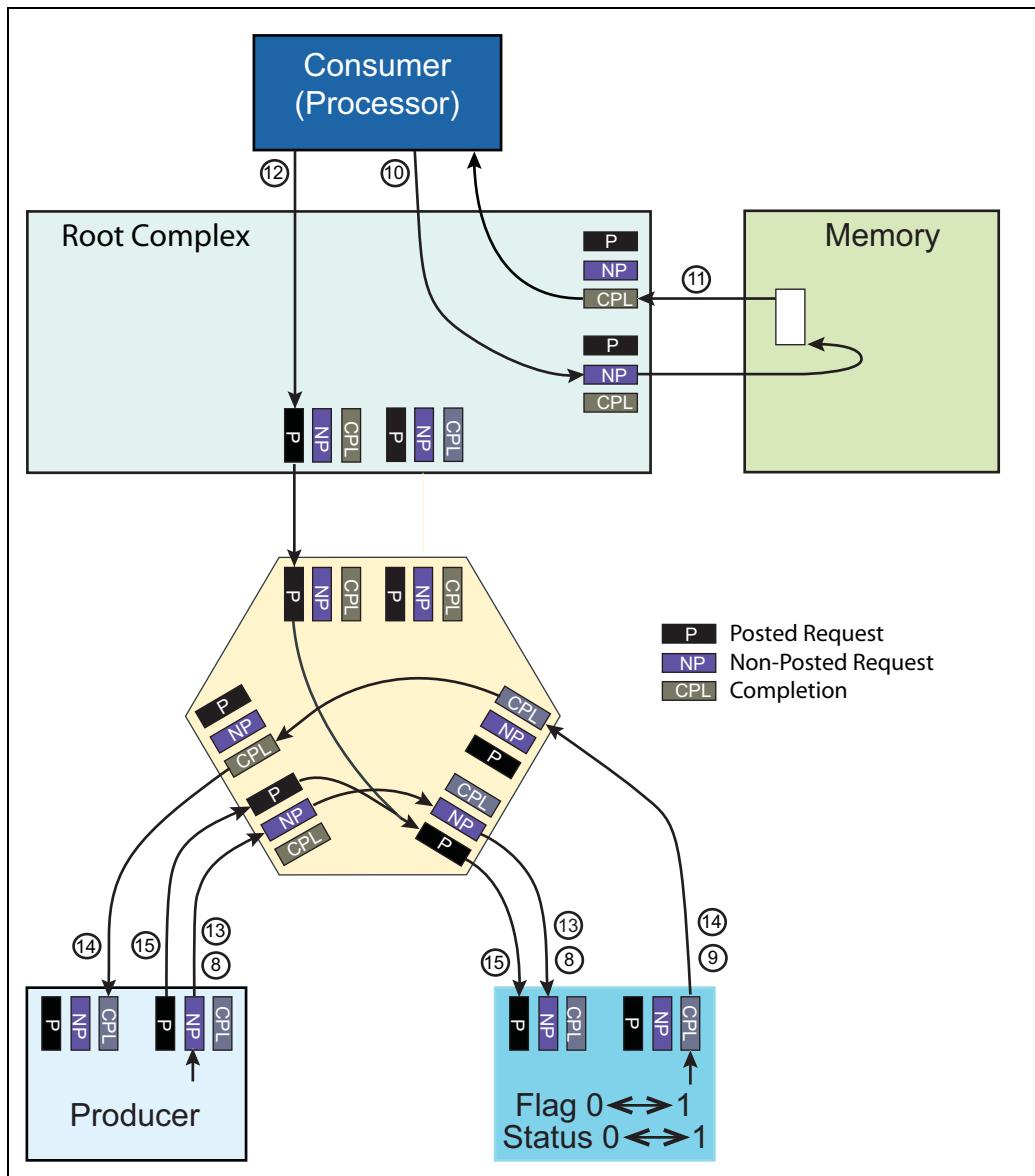
# Chapter 8: Transaction Ordering

Figure 8-2: Producer/Consumer Sequence Example — Part 1



# PCI Express 3.0 Technology

Figure 8-3: Producer/Consumer Sequence Example — Part 2



### Producer/Consumer Sequence — Errors

The previous example was handled correctly without a discussion of the ordering rules; however it may have been apparent that race conditions can cause the Producer/Consumer sequence to fail. Figure 8-4 on page 296 illustrates a simple sequence to demonstrate one of several problems that can arise without ordering rules being enforced. Refer to Figure Figure 8-4 on page 296 during the following discussion.

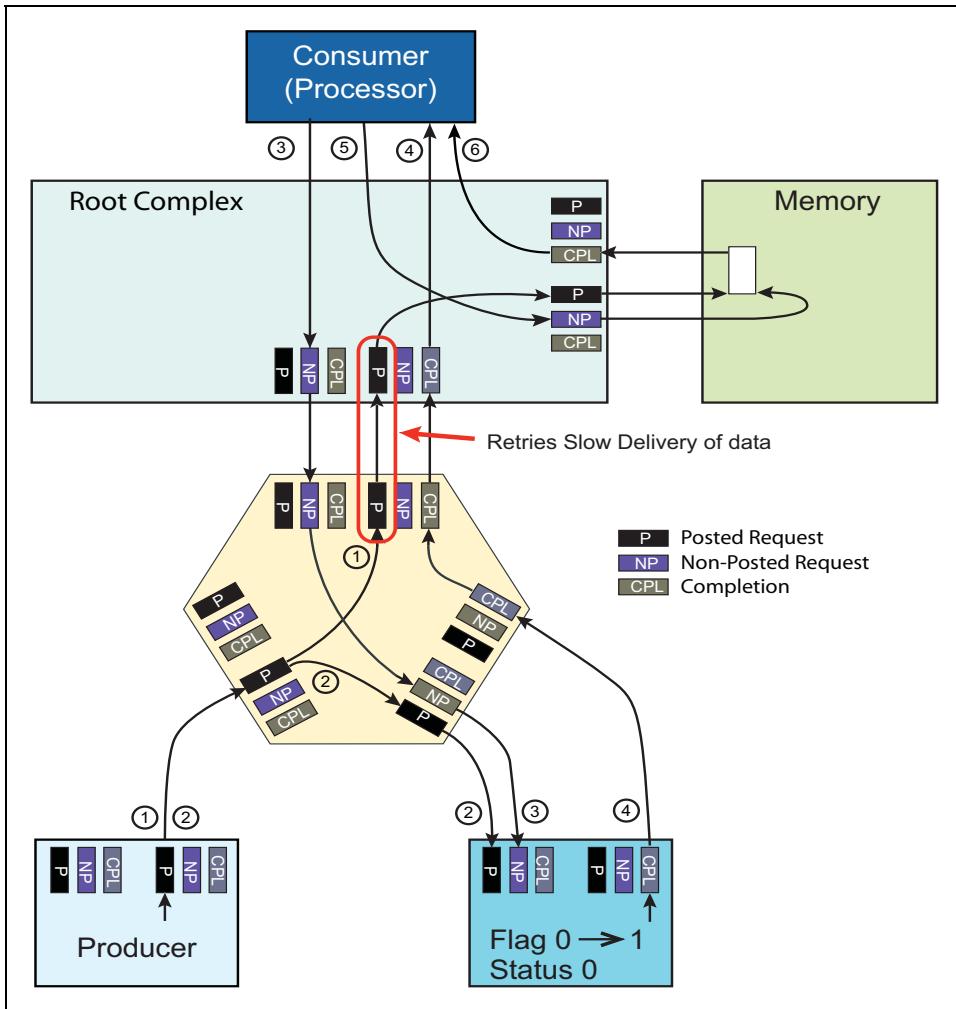
1. Producer performs a Memory Write request (Posted Request) to the memory buffer. Let us assume that the memory write data is temporarily stuck in the Switch upstream port Posted Flow Control buffer.
2. The Producer sends a Memory Write Transaction (Posted Request) to update the Flag to 1.
3. The Consumer initiates a Memory Read Request (Non-Posted Request) to check if the Flag has been set to 1.
4. The contents of the Flag is returned to the Consumer via a Completion.
5. Knowing that data has been delivered to memory, the Consumer performs a memory read request to fetch the data. However, the Consumer is unaware that the data is temporarily stuck in a Posted Flow Control buffer due to lack of flow control credits associated with the link between the upstream switch port and the Root Complex. Consequently, the Consumer receives old data when the Completion is returned to the Consumer.

The problem is avoided with ordering rules supported by virtual PCI bridges within the topology. In this example, when the Consumer performed the Memory Read transaction in steps 3 and 4, the Virtual PCI bridge at the upstream switch port should not allow the contents of the flag (Completion 4) to be forwarded ahead of the previously posted data.

# PCI Express 3.0 Technology

---

Figure 8-4: Producer/Consumer Sequence with Error



---

## Relaxed Ordering

PCI Express supports the Relaxed Ordering (RO) mechanism added for PCI-X. RO allows switches in the path between the Requester and Completer to reorder some transactions when doing so would improve performance.

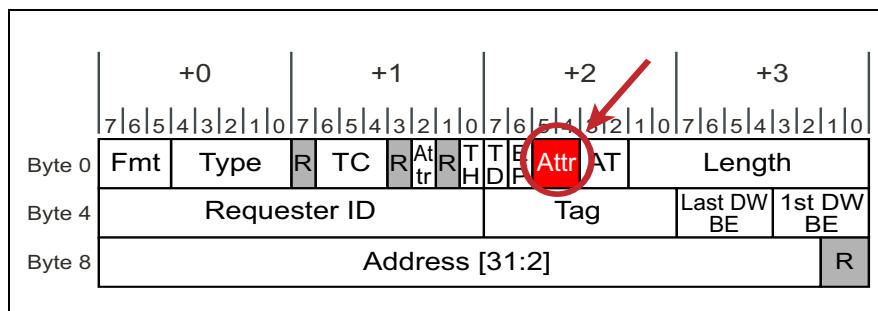
## Chapter 8: Transaction Ordering

---

The ordering rules that support the Producer/Consumer model may result in transactions being blocked in cases when they're unrelated to any Producer/Consumer transaction sequence. To alleviate this problem, a transaction can have its RO attribute bit set, indicating that software verifies it to be unrelated to other transactions, and that allows it to be re-ordered ahead of other transactions. For example, if a posted write is delayed because the target's buffer space is unavailable, then all subsequent transactions must wait until that finally resolves and the write is delivered. If a subsequent transaction was known by software to be unrelated to previous ones and the RO bit was set to show that, then it could be allowed to go before the write without risking a problem.

The RO bit (bit 5 of byte 2 of dword 0 in the TLP header as shown in Figure 8-5 on page 297) may be used by the device if its device driver has enabled it to do so. Request packets are then allowed to use this attribute as directed by software when it requests that a packet be sent. When switches or the Root Complex see a packet with this attribute bit set, they have permission to reorder it although it's not required that they should.

Figure 8-5: Relaxed Ordering Bit in a 32-bit Header



---

### RO Effects on Memory Writes and Messages

Switches and Root Complexes must observe the setting of the RO bit in transactions. Memory writes and Messages are both posted writes, both are received into the same Posted buffer, and both are subject to the same ordering requirements. When the RO bit is set, switches handle these transactions as follows:

- Switches are permitted to reorder memory write transactions just posted ahead of previously posted memory write transactions or message transactions. Similarly, message transactions just posted may be ordered ahead of

# PCI Express 3.0 Technology

---

previously posted memory write or message transactions. Switches must also forward the RO bit unmodified. The RO bit is ignored by PCI-X bridges, which always forward writes in order (there would be little purpose in allowing them to go out of order anyway; if one is blocked for some reason, the next will be blocked, too). Another difference is that message transactions had not been defined for PCI-X, either.

- The Root Complex is permitted to reorder posted write transactions (here it makes sense because the Root could write to different areas of memory so, if one area is busy it can write to a different one). Also, when receiving writes with RO set, the Root is permitted to write each byte to memory in any address order.

---

## RO Effects on Memory Read Transactions

All read transactions in PCI Express are handled as split transactions. When a device issues a memory read request with the RO bit set, the Completer returns the requested read data in a series of one or more split completion transactions, and uses the same RO setting as in the request. Switch behavior in this case is as follows:

1. A switch that receives a memory read with RO forwards the request in the order received, and must not reorder it ahead of memory write transactions that were previously posted. That guarantees that all write transactions moving in the direction of the read request are pushed ahead of the read. This is part of the Producer/Consumer example shown earlier, and software may depend on this flushing action for proper operation. The RO bit must not be modified by the switch.
2. When the Completer receives the memory read, it fetches the requested data and delivers one or more Completions that also have the RO bit set (its value is copied from the original request).
3. A switch receiving the Completions is allowed to re-order them ahead of previously posted memory writes moving in the direction of the Completion. If the writes were blocked (for example, due to flow control), then the Completions will be allowed to go ahead of them. Relaxed ordering in this case improves read performance. Table 8-2 summarizes the relaxed ordering behavior allowed by switches.

# Chapter 8: Transaction Ordering

---

Table 8-2: Transactions That Can Be Reordered Due to Relaxed Ordering

These Transactions with RO=1 Can Pass	These Transactions
Memory Write Request	Memory Write Request
Message Request	Memory Write Request
Memory Write Request	Message Request
Message Request	Message Request
Read Completion	Memory Write Request
Read Completion	Message Request

---

## Weak Ordering

Temporary transaction blocking can occur when strong ordering rules are rigorously enforced. Modifications that don't violate the Producer/Consumer programming model can eliminate some blocking conditions and improve link efficiency. Implementing the Weakly-Ordered model can alleviate this problem.

---

## Transaction Ordering and Flow Control

The motivation behind splitting VC buffers of a given number into flow controlled sub-buffers P, NP and CPL is because it simplifies processing of the transaction ordering rules once TLPs have been parsed or binned into their respective buffers. The transaction ordering processing logic then applies ordering rules between these three sub-buffers or to each sub-buffer.

Since TLPs are binned into their respective three sub-buffers in order to process transaction ordering rules, it is necessary to define the flow control mechanism between each virtual channel sub-buffer (P, NP, CPL) of neighboring ports at opposite ends of the Link. In fact, you may recall that there is an independent flow control mechanism between Header (Hdr) and Data (D) sub-buffers of each sub-buffer category (P, NP, CPL) of each virtual channel number.

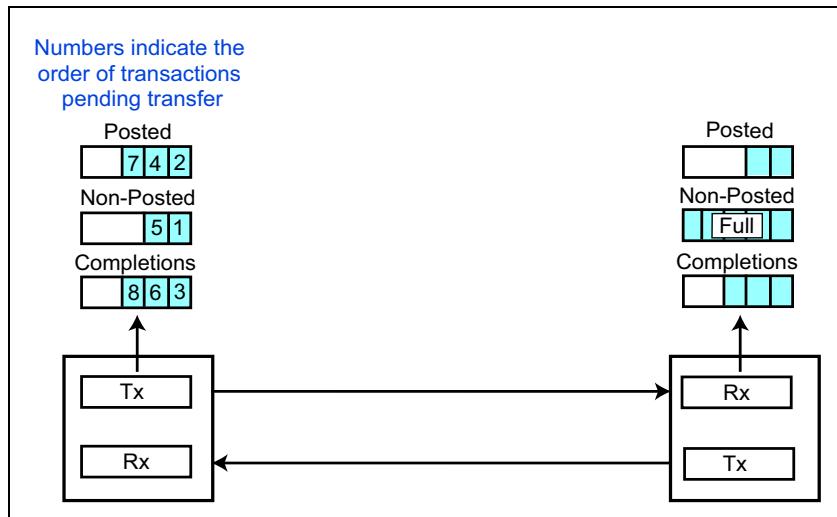
## Transaction Stalls

Strong ordering can result in instances where all transactions are blocked due to a single full receive buffer. For example, the ordering requirements for the Producer/Consumer model cannot be changed, but ordering for transactions that aren't part of that model can. To improve performance, let's consider a weakly-ordered scheme; one that puts the minimal requirements on transaction ordering.

This example depicts transmit and receive buffers associated with the delivery of transactions in a single direction for a single VC. Recall that each of the transaction types (Posted, Non-Posted, and Completions) have independent flow control within the same VC. The numbers in the transmit buffers show the order in which these transactions were issued, and the non-posted receive buffer is currently full. Consider the following sequence.

1. Transaction 1 (memory read) is the next transaction to send, but there aren't enough flow control credits so it must wait.
2. Transaction 2 (posted memory write) is the next subsequent transaction. If strong ordering is enforced, a memory write must not pass a previously queued read transaction.
3. This restriction applies to all subsequent transactions, too, with the result that they're all stalled until the first one finishes.

Figure 8-6: Strongly Ordered Example Results in Temporary Stall



## VC Buffers Offer an Advantage

Transaction ordering is managed within Virtual Channel buffers. These buffers are grouped into Posted, Non-Posted, and Completion transactions, and flow control is managed independently for each group. That makes weak ordering more useful because, as in our example, even if one buffer was full, others could still have space available.

---

## ID Based Ordering (IDO)

Another opportunity for optimizing ordering and improving performance is related to the nature of traffic streams. Packets from different requesters are very unlikely to have dependencies; after all, one device could hardly know when the other had finished certain steps based on ordering because they could have different paths to their shared resource. Bearing this in mind, the 2.1 revision of the PCIe spec introduced what is called ID-based Ordering to improve performance.

---

## The Solution

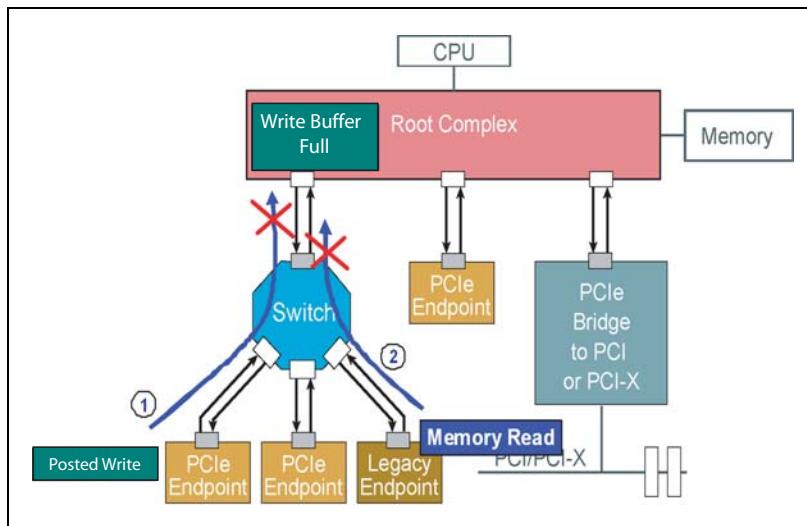
If the packet source isn't taken into account for transaction ordering then performance can suffer, as shown in Figure 8-7 on page 302. In the illustration, transaction 1 makes its way to the upstream port of the switch but is blocked from further progress by a buffer-full condition for that packet type in the Root port (which would be indicated by insufficient Flow Control credits). To use the spec terminology, packets from the same Requester are called a TLP stream. In this example, the path shown for Transaction 1 might include several TLPs as part of a TLP stream. Transaction 2 then arrives at the same egress port and is also blocked from moving forward because it must stay in order with Transaction 1. Since the packets came from different sources, (different TLP streams) this delay is almost certainly unnecessary; it's very unlikely they could have dependencies between them, but the normal ordering model doesn't take this into account. To get improved performance, we need another option.

The solution is simple: allow packets to be reordered if they don't use the same Requester ID (or Completer ID, for Completion packets). This optional capability allows software to enable a device to use IDO and a switch port can recognize that the packets are part of different TLP streams. This is done by setting the enable bits in Device Control 2 Register.

# PCI Express 3.0 Technology

---

Figure 8-7: Different Sources are Unlikely to Have Dependencies



---

## When to use IDO

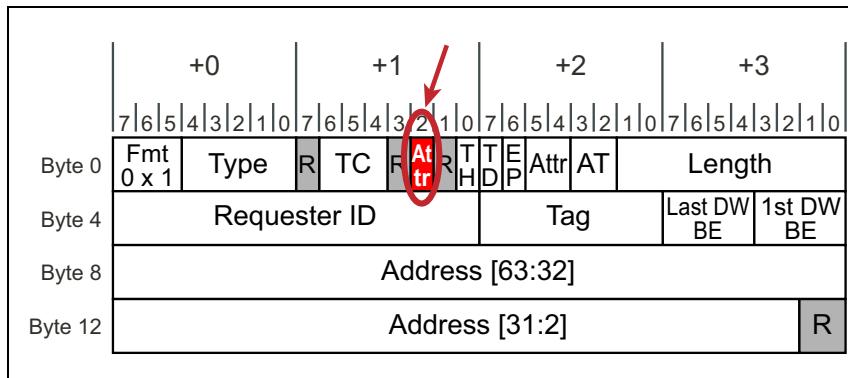
The spec highly recommends that both IDO and RO be used whenever safely possible. For example, it should be safe for Endpoints to use IDO for all TLPs when communicating directly with only one other entity, such as the Root Complex. On the other hand, it would not be safe to use it if the Endpoint is communicating with multiple agents. An example failure case for this from the spec begins with one device doing a DMA write to memory and then doing a peer-to-peer write to a flag in another device. When the second device receives the flag, it also initiates a DMA write to the same area of memory. Normally, the two DMA operations would stay in order, but with IDO that ordering can't be guaranteed because upstream devices will see them as coming from different device IDs. Similarly, it would not be safe to use RO with packets that are involved in control traffic.

For Completers, if IDO is enabled it's recommended that it be used for all Completions unless there is a specific reason not to do so.

## Software Control

Software can enable the use of IDO for Requests or Completions from a given port by setting the appropriate bits in its Device Control 2 Register. As with RO, there are no capability bits to let software find out what the device supports, just enable bits, so software would need to know by some other means that the device was capable of doing this. These bits enable the use of IDO for that packet type, but software must still decide whether each individual packet will have its IDO bit set. A new attribute bit in the header indicates whether a TLP is using IDO, as shown in Figure 8-8 on page 303. This brings up another related point: Completions normally inherit all the attribute bits of the Request that generated them, but this may not be true for IDO, since this can be enabled independently by the Completer. In other words, Completions may use IDO even if the Request that initiated them did not.

Figure 8-8: IDO Attribute in 64-bit Header



---

## Deadlock Avoidance

Because the PCI bus employs delayed transactions or because PCI Express memory read request may be blocked due to lack of flow control credits, several deadlock scenarios can develop. These deadlock avoidance rules are included in PCI Express ordering to ensure that no deadlocks occur regardless of topology. Adhering to the ordering rules prevent problems when boundary conditions develop due to unanticipated topologies (e.g., two PCI Express to PCI bridges connected across the PCI Express fabric). Refer to the MindShare book entitled *PCI System Architecture, Fourth Edition* (published by Addison-Wesley) for a detailed explanation of the scenarios that are the basis for the PCI Express

## **PCI Express 3.0 Technology**

---

ordering rules related to deadlock avoidance. Table 8-1 on page 289 lists the deadlock avoidance ordering rules which are identified as entries A3, A4, D3, D4 and A5b. Note that avoiding the deadlocks involves “Yes” entries in each of these 5 cases. If blocking occurs due to lack of flow control credits associated with the Non-Posted Request buffer identified in column 3 or 4, the Posted Requests associated with row A or the Completions associated with row D must be moved ahead of the Non-Posted Requests specified in the column 3 or 4 where the “Yes” entry exists. Note also that the “Yes” entry in A5b applies only to PCI Express to PCI or PCI-X Bridges.

Essentially, this deadlock avoidance rule can be summarized as “later arriving Memory Write Requests or Completions must be allowed to pass earlier blocked Non-Posted Requests otherwise a deadlock could result”.

# 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.

---

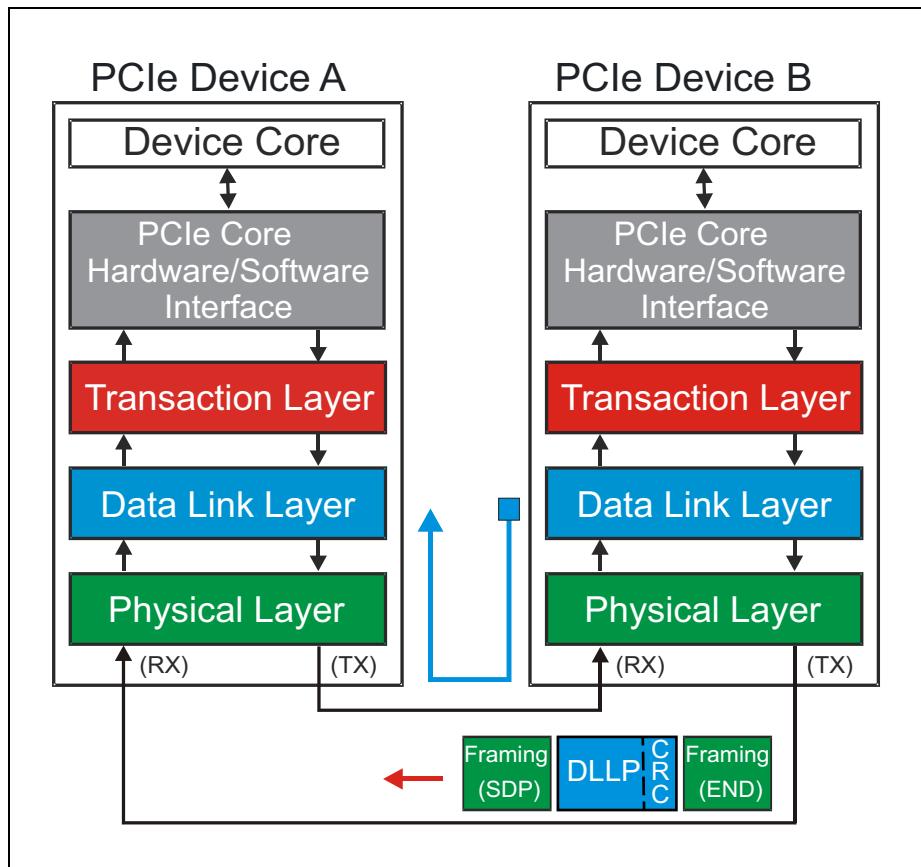
## General

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.

# PCI Express Technology

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

---

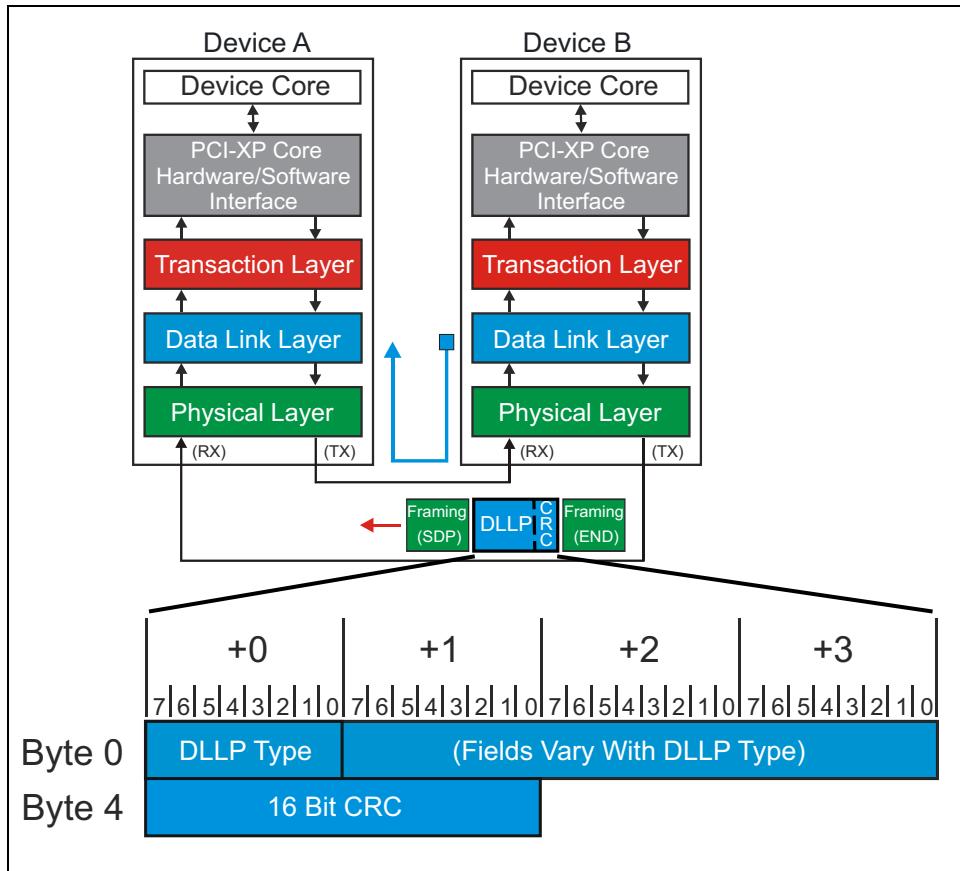
### General

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.

# PCI Express Technology

---

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.

# Chapter 9: DLLP Elements

---

3. If 8b/10b encoding is in use, a Start of DLLP (SDP) control symbol and an End Good (END) control symbol are added to the beginning and end of the packet. As usual, before transmission the Physical Layer encodes the bytes into 10-bit symbols for transmission.
4. In Gen3 mode, when 128b/130b encoding is in use, a 2-byte SDP Token is added to the front of the packet to create the 8-byte packet and there is no END symbol or token.

Note that there is never a data payload with a DLLP; all the information is carried in the core four bytes of the packet.

---

## DLLP Packet Types

There are four groups of DLLPs defined that deal with Ack/Nak, Power Management, and Flow Control, along with one Vendor Specific version. Some of these have several variants, and Table 9-1 on page 311 summarizes each variant as well as their *DLLP Type* field encoding.

Table 9-1: DLLP Types

DLLP Type	Type Field Encoding	Purpose
Ack (TLP Acknowledge)	0000 0000b	TLP transmission integrity
Nak (TLP Negative Acknowledge)	0001 0000b	TLP transmission integrity
PM_Enter_L1	0010 0000b	Power Management
PM_Enter_L23	0010 0001b	Power Management
PM_Active_State_Request_L1	0010 0011b	Power Management
PM_Request_Ack	0010 0100b	Power Management
Vendor Specific	0011 0000b	Vendor Defined
InitFC1-P	0100 0xxxb	TLP Flow Control (xxx = VC number)
InitFC1-NP	0101 0xxxb	TLP Flow Control

# PCI Express Technology

---

Table 9-1: DLLP Types (Continued)

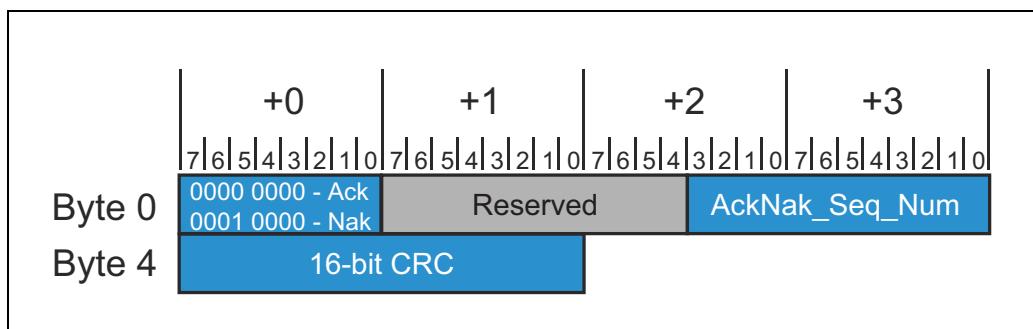
DLLP Type	Type Field Encoding	Purpose
InitFC1-Cpl	0110 0xxxb	TLP Flow Control
InitFC2-P	1100 0xxxb	TLP Flow Control
InitFC2-NP	1101 0xxxb	TLP Flow Control
InitFC2-Cpl	1110 0xxxb	TLP Flow Control
UpdateFC-P	1000 0xxxb	TLP Flow Control
UpdateFC-NP	1001 0xxxb	TLP Flow Control
UpdateFC-Cpl	1010 0xxxb	TLP Flow Control
Reserved	Others	Reserved

---

## Ack/Nak DLLP Format

The format of the DLLP used by a device to Ack (acknowledge) or Nak (negatively acknowledge) the receipt of a TLP is illustrated in Figure 9-3, and its fields are described in “Ack/Nak DLLP Fields” on page 313. For more discussion on how these are used to handle the Ack/Nak protocol, refer to Chapter 10, entitled “Ack/Nak Protocol,” on page 317.

Figure 9-3: Ack Or Nak DLLP Format



## Chapter 9: DLLP Elements

---

Table 9-2: Ack/Nak DLLP Fields

Field Name	Header Byte/Bit	DLLP Function
DLLP Type	Byte 0, [7:0]	Indicates the type of DLLP: <ul style="list-style-type: none"><li>• 0000 0000b = Ack</li><li>• 0001 0000b = Nak</li></ul>
AckNak_Seq_Num	Byte 2, [3:0] Byte 3, [7:0]	If a good TLP was received: <ul style="list-style-type: none"><li>• If incoming Sequence Number = NEXT_RCV_SEQ (matched what was expected), schedule Ack DLLP with that number.</li><li>• If incoming Sequence Number was earlier than NEXT_RCV_SEQ count (a duplicate TLP was received), schedule Ack DLLP with NEXT_RCV_SEQ - 1 (effectively, this is the number of the last good TLP).</li></ul> For a TLP received with a problem: <ul style="list-style-type: none"><li>• If the TLP had an error, or its Sequence Number was higher than NEXT_RCV_SEQ, schedule a Nak DLLP with NEXT_RCV_SEQ - 1.</li></ul>
16-bit CRC	Byte 4, [7:0] Byte 5, [7:0]	This 16-bit CRC protects the contents of this DLLP. Calculation is based on Bytes 0-3 of the Ack/Nak.

---

## Power Management DLLP Format

Power management DLLP information is shown in Figure 9-4, and its fields are described in Table 9-3 on page 314. To learn more about the use of these packets in power management, refer to Chapter 16, entitled "Power Management," on page 703.

# PCI Express Technology

---

Figure 9-4: Power Management DLLP Format

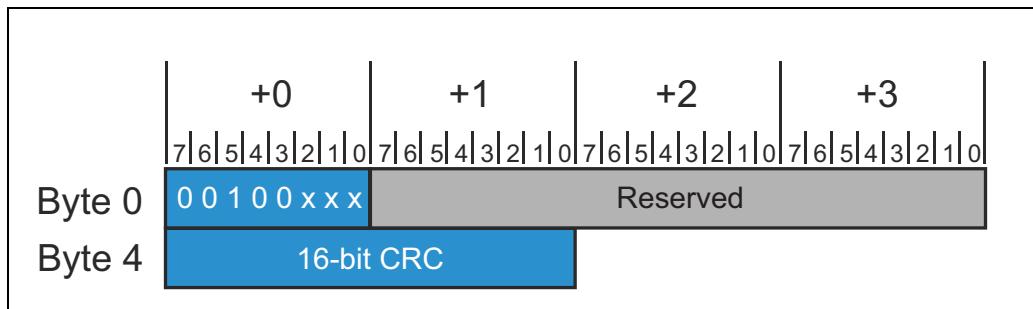


Table 9-3: Power Management DLLP Fields

Field Name	Header Byte/Bit	DLLP Function
DLLP Type	Byte 0, [7:0]	Indicates DLLP type. For Power Management DLLPs: 0010 0000b = PM_Enter_L1 0010 0001b = PM_Enter_L23 0010 0011b = PM_Active_State_Request_L1 0010 0100b = PM_Request_Ack
16-bit CRC	Byte 4, [7:0] Byte 5, [7:0]	A 16-Bit CRC used to protect DLLP contents. Calculation is based on Bytes 0-3, regardless of whether fields are used.

---

## Flow Control DLLP Format

Like many other serial transport buses, PCIe improves transport efficiency by using a credit-based flow control scheme. This topic is covered in detail in Chapter 6, entitled "Flow Control," on page 215. DLLPs are used to communicate flow control credit information. A variety of different DLLPs initialize flow control credits. Another category of update DLLPs are used to manage the runtime credit management as receiver buffer space is recovered. There are two Flow Control Initialization DLLPs called InitFC1 and InitFC2, and one Flow Control Update DLLP called UpdateFC.

The packet format for all three variants is illustrated in Figure 9-5 on page 315, while Table 9-4 on page 315 describes the fields contained in it.

## Chapter 9: DLLP Elements

Figure 9-5: Flow Control DLLP Format

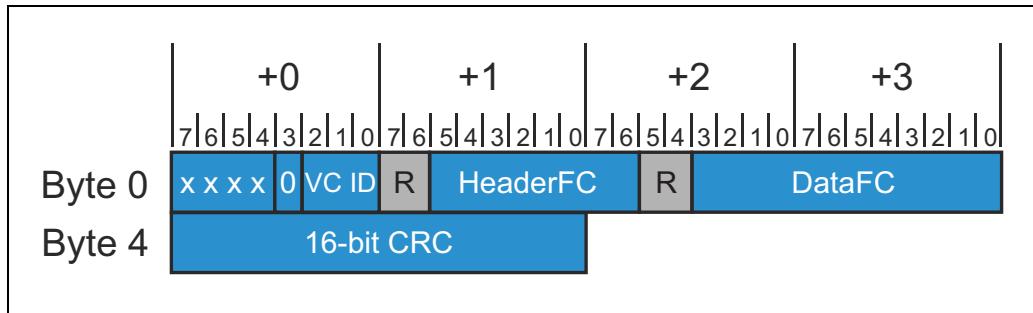


Table 9-4: Flow Control DLLP Fields

Field Name	Header Byte/Bit	DLLP Function
DLLP Type	Byte 0, [7:4]	This code indicates the type of FC DLLP: 0100b = InitFC1-P (Posted Requests) 0101b = InitFC1-NP (Non-Posted Requests) 0110b = InitFC1-Cpl (Completions) 0101b = InitFC2-P (Posted Requests) 1101b = InitFC2-NP (Non-Posted Requests) 1110b = InitFC2-Cpl (Completions) 1000b = UpdateFC-P (Posted Requests) 1001b = UpdateFC-NP (Non-Posted Requests) 1010b = UpdateFC-Cpl (Completions)
	Byte 0, [3]	Must be 0b as part of flow control encoding.
	Byte 0, [2:0]	VC ID. Indicates the Virtual Channel (VC 0-7) to be updated with these credits.
HdrFC	Byte 1, [5:0] Byte 2, [7:6]	Contains the credit count for header storage for the specified Virtual Channel. Each credit represents space for 1 header + the optional TLP Digest (ECRC).
DataFC	Byte 2, [3:0] Byte 3, [7:0]	Contains the credit count for data storage for the specified Virtual Channel. Each credit represents 16 bytes.

# PCI Express Technology

---

Table 9-4: Flow Control DLLP Fields (Continued)

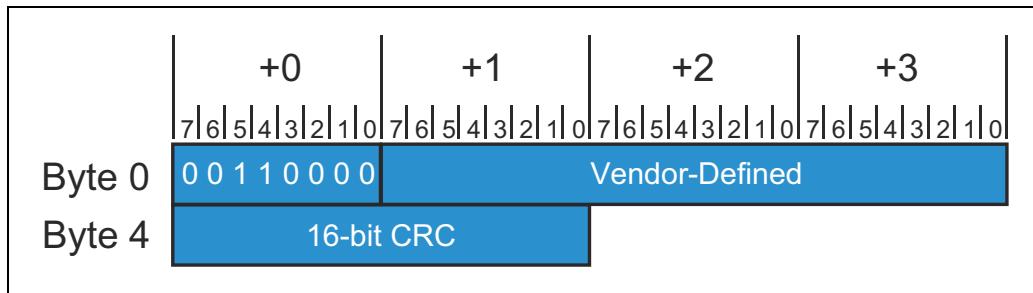
Field Name	Header Byte/Bit	DLLP Function
16-bit CRC	Byte 4, [7:0] Byte 5, [7:0]	A 16-Bit CRC that protects the contents of this DLLP. Calculation is based on Bytes 0-3, regardless of whether all fields are used.

---

## Vendor-Specific DLLP Format

The last defined DLLP type is used for vendor specific purposes. Therefore only the DLLP Type field is defined by the spec (0011 0000b), leaving the remaining contents available for vendor-defined use.

Figure 9-6: Vendor-Specific DLLP Format



---

# 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.

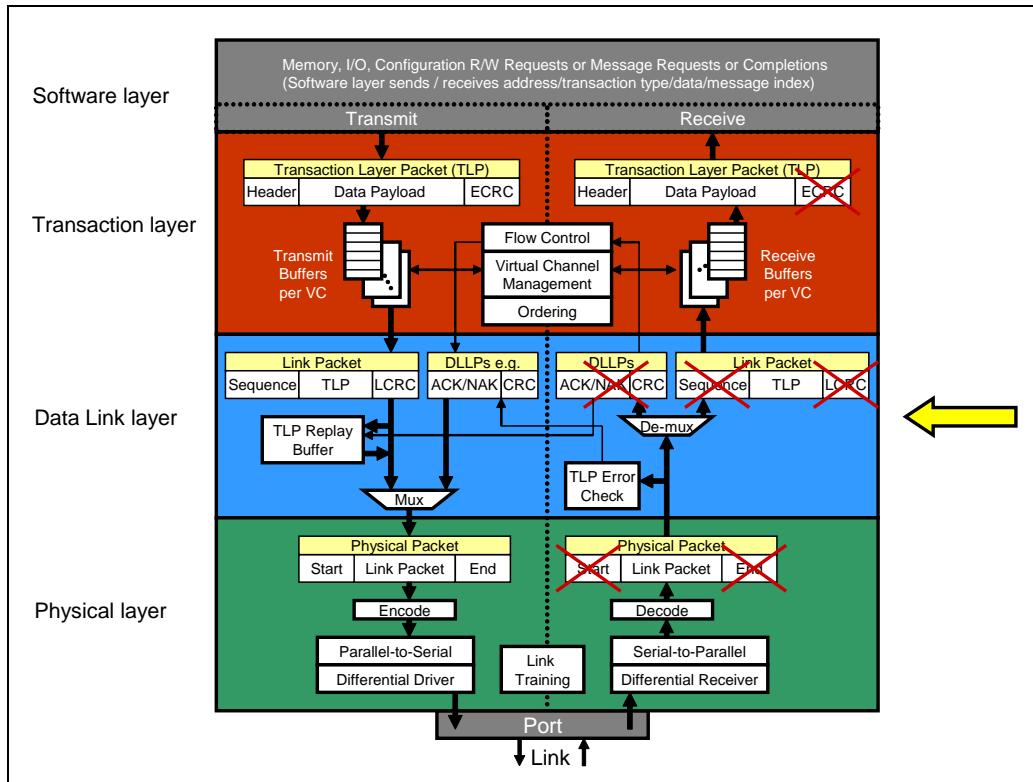
---

## 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.

# PCI Express Technology

Figure 10-1: Data Link Layer



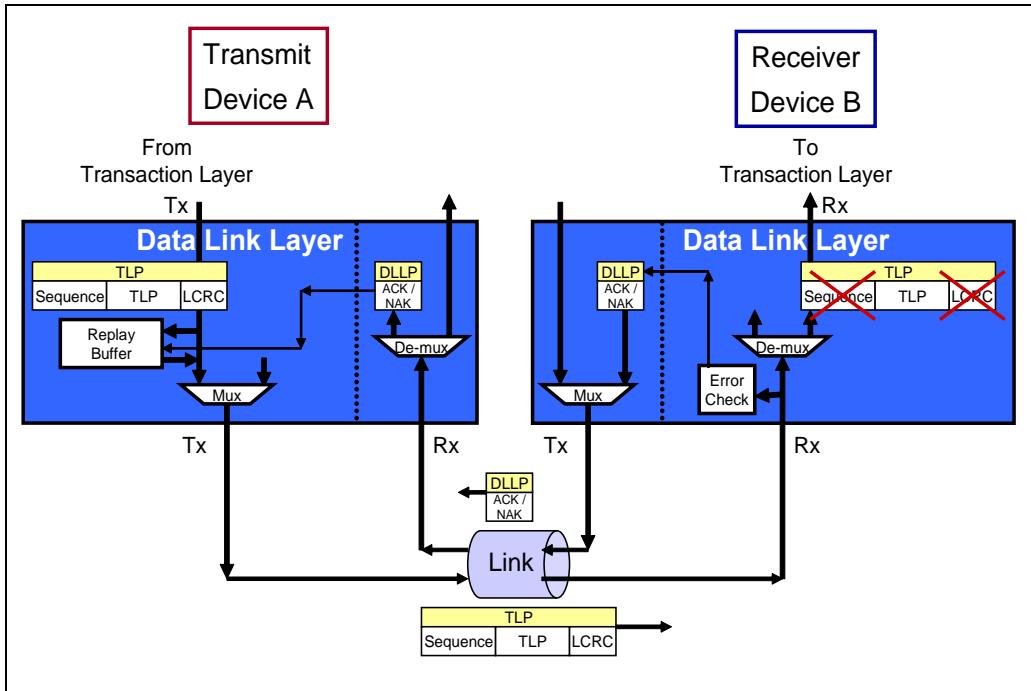
To facilitate this goal, an error detection code called an LCRC (Link Cyclic Redundancy Code) is added to each TLP. The first step in error checking is simply to verify that this code still evaluates correctly at the receiver. If each packet is given a unique incremental Sequence Number as well, then 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

# Chapter 10: Ack/Nak Protocol

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



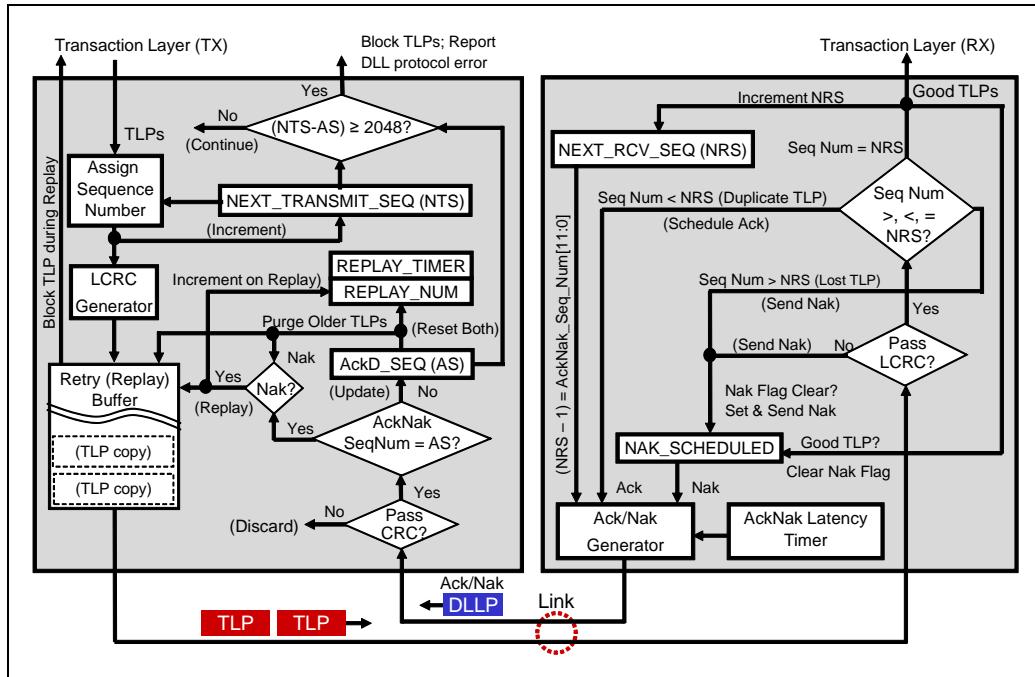
Since both the sending and receiving devices in the protocol have both a transmit and a receive side, this chapter will use the terms:

- **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.

## NEXT\_TRANSMIT\_SEQ Counter

This counter generates the Sequence Number that will be assigned to the next incoming TLP. It's a 12-bit counter that is initialized to 0 at reset or when the Link Layer reports DL\_Down (Link Layer is inactive). Since it increments continuously with each TLP and only counts forward, the counter eventually reaches its max value of 4095 and rolls over to 0 as it continues to count.

This Sequence Number assigned to the TLP will be used in the Ack or Nak sent by the receiver to reference this TLP in the Replay Buffer. One might think that such a large counter means that a large number of unacknowledged TLPs could be in flight, but in practice this is very unlikely. The main reason is that the receiver has a requirement to send an Ack back for successfully received TLPs within a certain amount of time. That amount of time is discussed in detail in "AckNak\_LATENCY\_TIMER" on page 328, but is typically only long enough to transmit a few max sized packets.

## LCRC Generator

This block generates a 32-bit CRC (Cyclic Redundancy Check) code based on the header and data to be sent and adds it to the end of the outgoing packet to facilitate error detection. The name is derived from the fact that this *check code* (calculated from the packet to be sent) is *redundant* (adds no information), and is derived from *cyclic codes*. Although a CRC doesn't supply enough information for the Receiver to do automatic error correction the way ECC (Error Correcting Code) methods can, it does provide robust error detection. CRCs are commonly used in serial transports because they're easy to implement in hardware, and because they're good at detecting burst errors: a string of incorrect bits. Since this is more likely to happen in a serial design than a parallel model, it helps explain why a CRC is a good choice for error detection in serial transports. The CRC code is calculated using all fields of the TLP, including the Sequence Number. The receiver will make the same calculation and compare its result to the LCRC field in the TLP. If they don't match, an error is detected in the Receiver's Link Layer.

## Replay Buffer

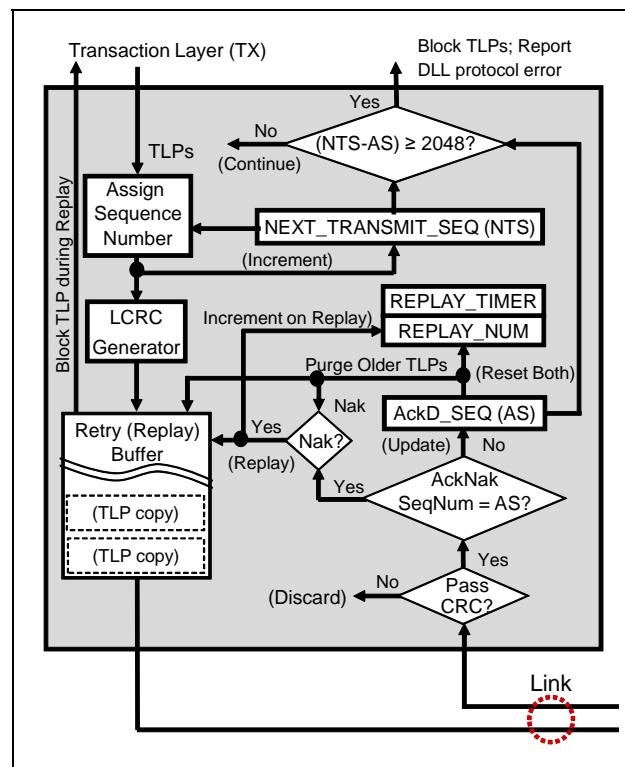
The replay buffer, or retry buffer, stores TLPs, including the Sequence Number and LCRC, in the order of their transmission. When the transmitter receives an Ack indicating that TLPs have reached the receiver successfully, it purges from the Replay Buffer those TLPs whose Sequence Number is equal to or earlier than the number in the Ack. In this way, the design allows one Ack to represent several successful TLPs, reducing the number of Acks that must be sent. Since the packets must always be seen in order, then if an Ack is received with a

# PCI Express Technology

Sequence Number of 7, then not only was TLP 7 received successfully, but all the packets before it must also have been received successfully, so there is no reason to keep a copy of them in the replay buffer.

If a Nak is received, the Sequence Number in the Nak still indicates the last *good* packet received. So even receiving a Nak can cause the transmitter to purge TLPs from the replay buffer. However, because it is a Nak, it means that something was not received successfully at the receiver, so after purging all the acknowledged TLPs, the transmitter must replay everything still in the replay buffer in order. For example, if a Nak is received with a Sequence Number of 9, then packet 9 and all prior packets are purged from the replay buffer, because the receiver acknowledged that they have been successfully received. However, because it is a Nak, the transmitter must then replay all the remaining TLPs in the replay buffer in order, starting with packet 10.

Figure 10-4: Transmitter Elements Associated with the Ack/Nak Protocol



## **REPLAY\_TIMER Count**

This timer is effectively a watchdog timer. It makes sure that the transmitter is receiving Ack/Nak packets for TLPs that have been transmitted. If this timer expires, it means that the transmitter has sent one or more TLPs that it has not received an acknowledgement for in the expected time frame. The fix is to retransmit everything in the replay buffer and restart the REPLAY\_TIMER.

This timer is running anytime a TLP has been transmitted but not yet acknowledged. If the REPLAY\_TIMER is not currently running, it is started when the last Symbol of any TLP is transmitted. If the timer is already running, then sending additional TLPs does not reset the timer value. When an Ack or Nak is received that acknowledges TLPs in the replay buffer, the timer resets back to 0, and if there are still TLPs in the replay buffer (TLPs that have been transmitted, but not yet acknowledged), it immediately starts counting again. However, if an Ack is received that acknowledges the last TLP in the replay buffer, meaning the replay buffer is now empty, the REPLAY\_TIMER resets to 0 but does not count. It will not begin counting again until the last Symbol of the next TLP is transmitted.

## **REPLAY\_NUM Count**

This 2-bit counter tracks the number of replay attempts after reception of a Nak or a REPLAY\_TIMER time-out. When the REPLAY\_NUM count rolls over from 11b to 00b (indicating 4 failed attempts to deliver the same set of TLPs), the Data Link Layer automatically forces the Physical Layer to retrain the Link (LTSSM goes to the Recovery state). When re-training is finished, it will attempt to send the failed TLPs again. The REPLAY\_NUM counter is initialized to 00b at reset, or when the Link Layer is inactive. It is also reset whenever an Ack DLLP is received with a Sequence Number that is more recent than the last one seen, meaning forward progress is being made.

## **ACKD\_SEQ Register**

This 12-bit register stores the Sequence Number of the most recently received Ack or Nak. It is initialized to all 1s at reset, or when the Data Link Layer is inactive. This register is updated with the AckNak\_Seq\_Num [11:0] field of a received Ack or Nak. The ACKD\_SEQ count is compared with the Sequence Number in the last received Ack or Nak to check for forward progress. If the latest Ack/Nak had a Sequence Number later than the ACKD\_SEQ register, then we're making forward progress.

As an aside, we use the term “later Sequence Number” to account for the fact that, like most counters in PCIe, the Sequence Number counters only count forward, meaning that they’ll eventually roll over back to zero. Technically, a later number would mean a numerically higher value, but we have to remember that when the counter reaches 4095 (it’s a 12-bit counter), the next higher number will be zero. This wrap-around effect will be easier to see in the examples later, as in “Ack/Nak Examples” on page 331.

As shown in Figure 10-4 on page 322, when an Ack or Nak makes forward progress it causes TLPs with Sequence Numbers equal to or older than the value in the DLLP to be purged out of the Replay Buffer. It also resets both the REPLAY\_TIMER and the REPLAY\_NUM count. If no forward progress is made, no TLPs can be purged so we only check to see if it’s a Nak that would necessitate a replay.

This is a good place to mention a potential problem with the counters: the number of TLPs sent might theoretically become much larger than the number that have been acknowledged by the receiver. As mentioned earlier, this is very unlikely; it’s only mentioned here for completeness. The problem is basically the same as it for the Flow Control counters (see “Stage 3 — Counters Roll Over” on page 234) and has the same solution: the NEXT\_TRANSMIT\_SEQ and ACKD\_SEQ counters are never allowed to be separated by more than half their total count value. If a large number of TLPs are sent without acknowledgement so that the NEXT\_TRANSMIT\_SEQ count value is later than ACKD\_SEQ count by 2048, no more TLPs will be accepted from the Transaction Layer until this is resolved by receiving more Acks. If the difference between the Sequence Number sent and the acknowledged count ever did exceed half the maximum count value, a Data Link Layer protocol error would be reported. (For more on error reporting, see “Data Link Layer Errors” on page 655.)

## DLLP CRC Check

This block checks for errors in the 16-bit CRC of DLLPs. If an error is detected, the DLLP is discarded and a Correctable Error may be reported, if enabled. No further action is taken because there is no mechanism to replay or correct failed DLLPs. Instead, we simply wait for the next successful Ack/Nak, which will get the counters back up-to-date and allow normal operation to continue.

---

## Receiver Elements

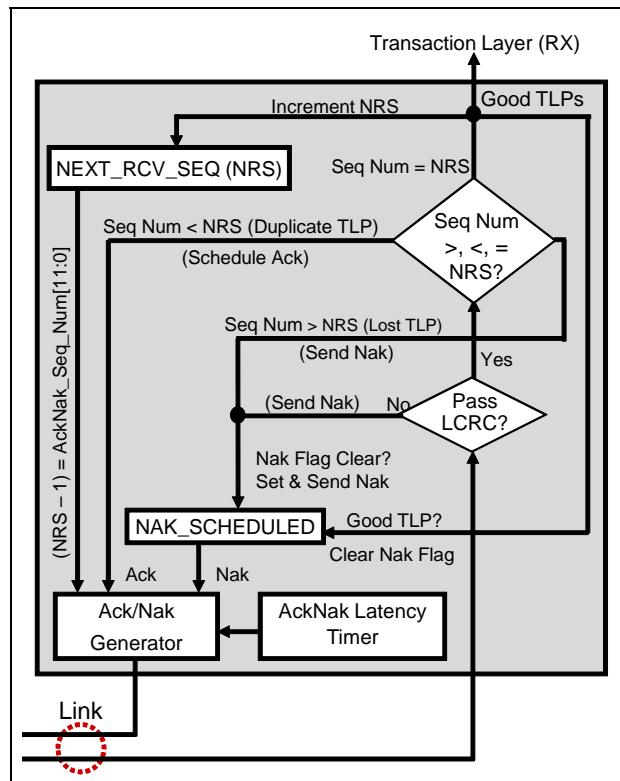
Incoming TLPs are first checked for LCRC errors and then for Sequence Numbers. If there are no errors, the TLP is forwarded to the receiver’s Transaction

# Chapter 10: Ack/Nak Protocol

Layer. If there are errors, the TLP is discarded and a Nak will be scheduled unless there was already a Nak outstanding.

Figure 10-5 on page 325 illustrates the receiver Data Link Layer elements associated with processing of inbound TLPs and outbound Ack/Nak DLLPs.

Figure 10-5: Receiver Elements Associated with the Ack/Nak Protocol



## LCRC Error Check

This block checks for transmission errors in the received TLP by verifying the 32-bit LCRC. This block calculates an LCRC value based on the received bits of the TLP and then compares the calculated LCRC to the received LCRC. If they match, then all the bits of the packet were received exactly as they were transmitted. If it doesn't match, then there was a bit error in the TLP so it gets dropped and a Nak will be sent to get a replay of that packet and any TLPs sent after the bad packet.

## NEXT\_RCV\_SEQ Counter

The 12-bit NEXT\_RCV\_SEQ (Next Receive Sequence number) counter keeps track of the expected Sequence Number and is used to verify sequential packet reception. It's initialized to 0 at reset or when the Data Link Layer is inactive, and is incremented once for each good TLP forwarded to the Transaction Layer. TLPs that have errors or were nullified are not sent to the Transaction Layer and therefore don't increment this counter.

## Sequence Number Check

If the LCRC check was OK, the TLP's Sequence Number is checked against the expected count (the NRS number). As can be seen in Figure 10-5 on page 325, there are three possible outcomes of this check:

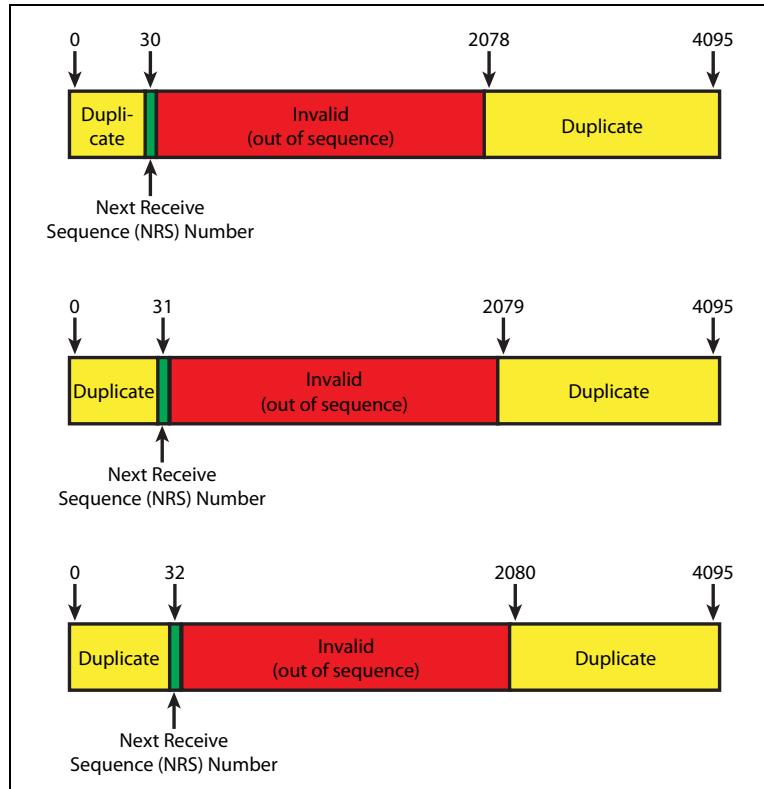
1. The TLP Sequence Number equals the NRS count (the number we're expecting). In this case, everything is good: the TLP is accepted and forwarded to the Transaction Layer and the NRS count is incremented. The Receiver schedules an Ack, but it doesn't have to be sent until the AckNak\_LATENCY\_TIMER expires. In the meantime, other good TLPs may be received, incrementing the NEXT\_RCV\_SEQ counter. Then, once the timer expires, a single Ack is sent with the Sequence Number of the last good TLP received (NRS - 1). That allows one Ack to represent several successful TLPs and reduces overhead, since a dedicated Ack is not required for every TLP.
2. If the TLP's Sequence Number is earlier than the NRS count (smaller than expected), this TLP has been seen before and is a duplicate. As long as the expected Sequence Number and received Sequence Number don't get separated by more than half the total count value (2048), this is not an error, but is seen as a duplicate, meaning the TLP has already been accepted earlier. In this case, the TLP is silently dropped (no Nak, no error reporting) and an Ack is sent with the Sequence Number of the last good TLP it received (NRS - 1). Why would this situation happen? The transmitter may not have received a transmitted Ack, so his REPLAY\_TIMER expired and he retransmitted everything in his Replay Buffer. By sending the transmitter an Ack with the Sequence Number of the last good packet we received, we're notifying him of the furthest progress we've made.
3. If the TLP's Sequence Number is a later Sequence Number than NEXT\_RCV\_SEQ count (larger than expected), then the Link Layer has missed a TLP. For example, if we're expecting Sequence Number 30 and the incoming TLP has Sequence Number 31 we know there's a problem. The numbers must be sequential and, since they aren't, one must have failed

# Chapter 10: Ack/Nak Protocol

and been dropped, as might happen at the Physical Layer. This out-of-order TLP is discarded, whether or not it had any other errors because we must accept TLPs in order, and a Nak will be sent if there wasn't one already outstanding.

The concept of the expected sequence number (NRS) incrementing as new TLPs are successfully received and seeing how that affects the sliding windows for the invalid range of sequence numbers and the duplicate range of sequence numbers can be seen in Figure 10-6.

Figure 10-6: Examples of Sequence Number Ranges



## NAK\_SCHEDULED Flag

This flag is set whenever the receiver schedules a Nak, and is cleared when the receiver successfully receives the TLP with the expected Sequence Number (NRS). The spec is clear that the receiver must not schedule additional Nak DLLPs while the NAK\_SCHEDULED flag remains set. The author's opinion is

# PCI Express Technology

---

that this is intended to prevent the possibility of an endless loop; a case in which the transmitter begins to replay some packets but the receiver sends another Nak before the replays finish and causes it to restart sending them again. Whatever the motivation, once a Nak has been sent there will be no more Naks forthcoming until the problem is resolved by successful receipt of the replayed TLP with the correct Sequence Number.

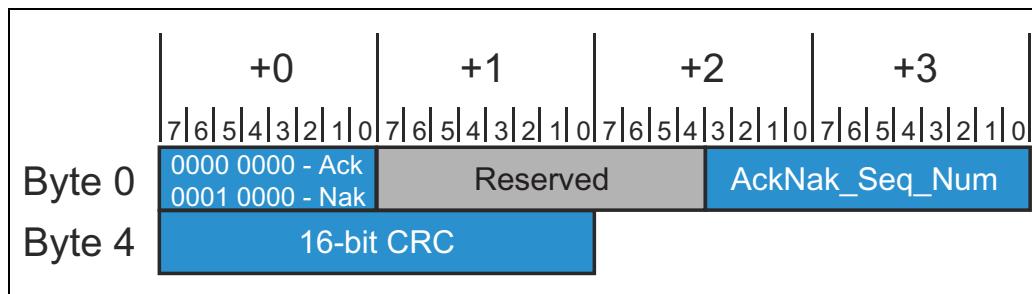
## AckNak\_LATENCY\_TIMER

This timer is running anytime a receiver successfully receives a TLP that it has not yet acknowledged. The receiver is required to send an Ack once the timer expires. The length of time the AckNak Latency Timer runs is dictated by the spec (see “AckNak\_LATENCY\_TIMER” on page 328) and determines how long a receiver can coalesce Acks. Once the AckNak Latency Timer expires, an Ack with sequence number NRS-1 is generated and sent which indicates the last good packet it received. This timer is reset whenever an Ack or Nak are sent and it only restarts once a new good TLP is received.

## Ack/Nak Generator

Ack or Nak DLLPs are scheduled by the error checking blocks and contain a 12-bit AckNak\_Seq\_Num field as illustrated in Figure 10-7 on page 328. It calculates this number by subtracting one from the NRS count, which results in reporting the last good Sequence Number received. That’s because a good TLP received increments NRS before scheduling the Ack, while a failed TLP just schedules a Nak without incrementing NRS. This method makes it easier to handle failed packets because the error in the TLP might have been in the Sequence Number, so that number can’t be used in the Nak. Instead, it uses the number of the last good TLP; what we’re expecting minus one. The only case where this value doesn’t represent the last good TLP is for the first TLP after a reset. If that first TLP, using Sequence Number 0, fails, the resulting Nak will have an AckNak\_Seq\_Num value of zero minus one which results in all 1’s.

Figure 10-7: Ack Or Nak DLLP Format



# Chapter 10: Ack/Nak Protocol

---

Table 10-1: Ack or Nak DLLP Fields

Field Name	Header Byte/Bit	DLLP Function
DLLP Type	Byte 0, [7:0]	Indicates the type: <ul style="list-style-type: none"><li>• 0000 0000b = Ack</li><li>• 0001 0000b = Nak</li></ul>
AckNak_Seq_Num	Byte 2, [3:0] Byte 3, [7:0]	This value will always be NEXT_RCV_SEQ count - 1.
16-bit CRC	Byte 4, [7:0] Byte 5, [7:0]	16-bit CRC used to protect the contents of this DLLP.

---

## Ack/Nak Protocol Details

This section describes the detailed transmitter and receiver behavior in processing TLPs and Ack/Nak DLLPs. Several examples are used to demonstrate various cases that may occur.

---

## Transmitter Protocol Details

### Sequence Number

Referring back to Figure 10-4 on page 322, when TLPs are delivered by the Transaction Layer to the Link Layer, one of the first steps is to append a 12-bit Sequence Number. Keep in mind that the next incremental Sequence Number may actually be smaller, as will happen when the counter rolls over back to zero after it reaches a maximum value of 4095. Consequently, a value of zero can actually be ‘larger’ than a value of 4095, for example. It may help to think of the Sequence Number comparison as evaluating a ‘window’ of numbers that consistently moves upward and rolls over. To clarify this concept, such a count roll-over is used in several of the upcoming examples.

### 32-Bit LCRC

The transmitter also generates and appends a 32-bit LCRC (Link CRC) based on the TLP contents (Sequence Number, Header, Data Payload and ECRC).

## Replay (Retry) Buffer

**General.** Before a device transmits a TLP, it stores a copy of the TLP in the Replay Buffer. (*Note that the spec uses the term ‘Retry Buffer’ but in this book ‘Replay’ was chosen instead of ‘Retry’ to more clearly distinguish this mechanism from the old PCI Retry mechanism.*) Each buffer entry stores a complete TLP with all of its fields including the Sequence Number (12 bits wide, it occupies two bytes), Header (up to 16 bytes), an optional Data Payload (up to 4KB), an optional ECRC (four bytes) and the LCRC field (four bytes).

It is important to note that the spec describes the Replay Buffer in this fashion, but it is NOT a spec requirement that it be implemented this way. As long as your device can replay a sequence of TLPs if required, as defined by the spec, then how that is accomplished within a device is completely up to the designer. Having a Replay Buffer that behaves as described above is one way to accomplish this.

**Replay Buffer Sizing.** The spec writers chose not to specify the Replay Buffer size, leaving it as an optimization for the device designers. It should be made big enough to store TLPs that haven’t yet been acknowledged by Acknowledgments so that under normal operating conditions it doesn’t become full and stall new TLPs coming in from the Transaction Layer, but also small enough to keep the cost down. To determine the optimal buffer size, a designer will consider:

- Ack DLLP Latency from the receiver.
- Delays caused by the physical Link.
- Receiver L0s exit latency to L0. In other words, the buffer should be big enough to hold TLPs without stalling while the Link returns from the L0s state to L0.

When the transmitter receives an Ack, it purges TLPs from the Replay Buffer with Sequence Numbers equal to or earlier than the Sequence Number in the Ack (*normally this term would be ‘smaller than’ but the counter roll-over behavior will sometimes make that an incorrect evaluation, so the term ‘earlier than’ was chosen instead*). Similarly, when the transmitter receives a Nak, it still purges the Replay Buffer of TLPs with Sequence Numbers that are equal to or earlier than the Sequence Number that arrives in the Nak, but then it also replays (re-sends) TLPs of later Sequence Numbers (the remaining TLPs in the Replay Buffer).

## Transmitter's Response to an Ack DLLP

A single Ack returned by the receiver may acknowledge multiple TLPs; it isn't necessary that every TLP transmitted receive a dedicated Ack. The receiver can get multiple good TLPs and send one Ack with the Sequence Number of the last good TLP received. The transmitter's response to an Ack that makes forward progress (has a Sequence Number that is later than the last one seen) is to load the AckD\_SEQ register with the Sequence Number of the new Ack. It also resets the REPLAY\_NUM counter and REPLAY\_TIMER, and purges the Replay Buffer of all TLPs that were acknowledged by that Ack.

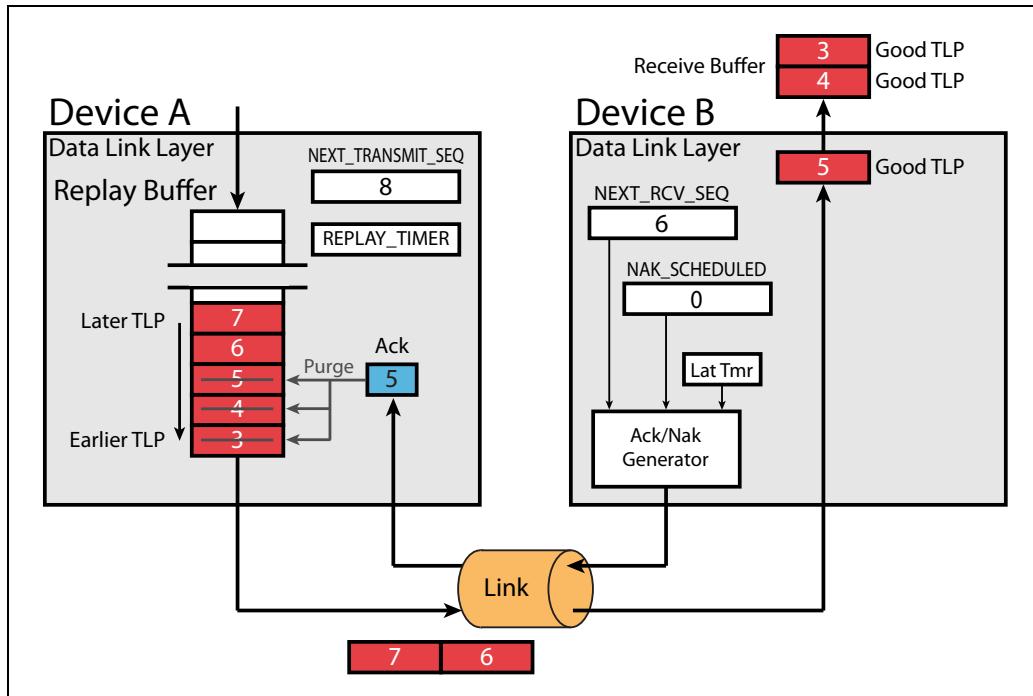
## Ack/Nak Examples

**Example 1.** Consider Figure 10-8 on page 332 for the following discussion.

1. Device A transmits TLPs with Sequence Numbers 3, 4, 5, 6, 7.
2. Device B successfully receives TLP 3 and increments its NEXT\_RCV\_SEQ counter from 3 to 4. Since Device B had previously acknowledged all successfully received TLPs, the AckNak\_LATENCY\_TIMER was not running. Having received TLP 3, Device B has now successfully received a TLP that it has not acknowledged, so the AckNak\_LATENCY\_TIMER is started (this is equivalent of scheduling an Ack).
3. Device B successfully receives TLPs 4 and 5 before the AckNak\_LATENCY\_TIMER expires. Receiving TLPs 4 and 5 does NOT reset the AckNak\_LATENCY\_TIMER.
4. Once the AckNak\_LATENCY\_TIMER expires, Device B sends a single Ack with the Sequence Number 5, the last good TLP received. The AckNak\_LATENCY\_TIMER is reset but does not restart until it successfully receives TLP 6.
5. Device A receives Ack 5, resets the REPLAY\_TIMER and REPLAY\_NUM counter, because forward progress is being made. And it purges TLPs from the Replay Buffer that have Sequence Numbers earlier than or equal to 5.
6. Once Device B receives TLPs 6 and 7 and its AckNak\_LATENCY\_TIMER expires again, it will send an Ack with a Sequence Number of 7 which will purge the last two TLPs in the Replay Buffer of Device A (according to this example).

# PCI Express Technology

Figure 10-8: Example 1 - Example of Ack

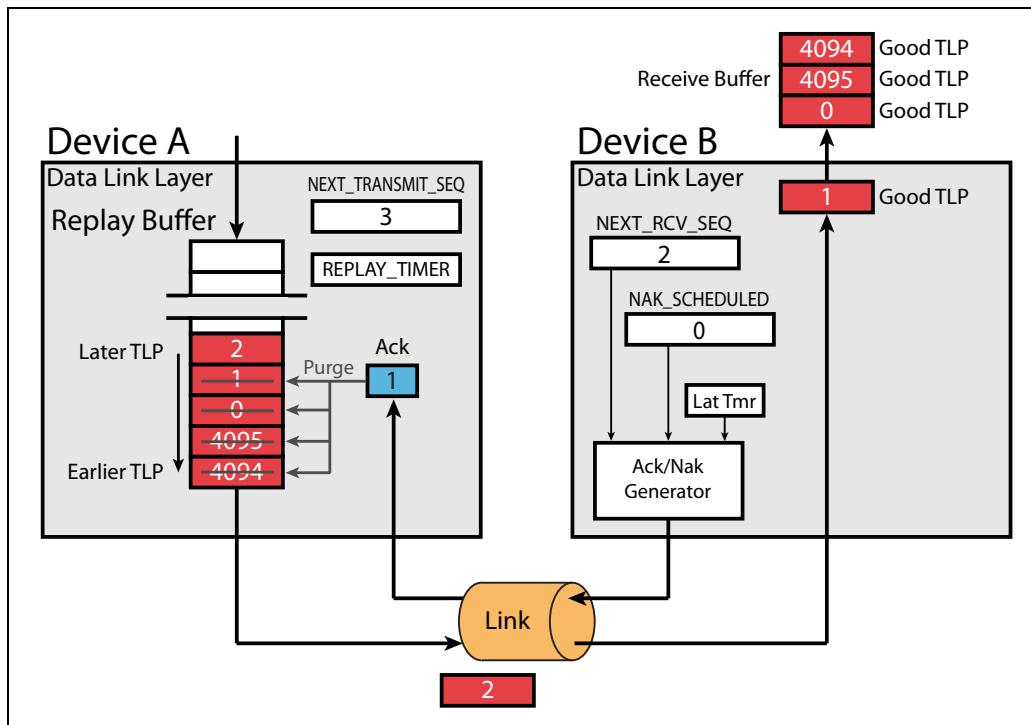


**Example 2.** This example is showing the exact same behavior as Example 1, but it is pointing out the rollover behavior for the Sequence Numbers, as show in Figure 10-9 on page 333.

1. Device A transmits TLPs with Sequence Numbers 4094, 4095, 0, 1, and 2 where TLP 4094 is the first TLP sent and TLP 2 is the last TLP sent in this example.
2. Device B successfully receives TLPs with Sequence Numbers 4094, 4095, 0, 1 in that order. Reception of TLP 4094 causes the AckNak\_LATENCY\_TIMER to start. TLPs 4095, 0 and 1 are received before the AckNak\_LATENCY\_TIMER expires. TLP 2 is still en route.
3. Because the AckNak\_LATENCY\_TIMER expires, Device B send an Ack with a Sequence Number of 1 to acknowledge receipt of TLP 1 and all prior TLPs (0, 4095 and 4094 in this example).
4. Device A successfully receives Ack 1, purges TLPs 4094, 4095, 0, and 1 from the Replay Buffer and resets the REPLAY\_TIMER and REPLAY\_NUM count.

# Chapter 10: Ack/Nak Protocol

Figure 10-9: Example 2 - Ack with Sequence Number Rollover



### **Transmitter's Response to a Nak**

A Nak indicates that a problem has occurred. When a transmitter receives one, it first purges from the Replay Buffer any TLPs with earlier or equal Sequence Numbers and then replays the remaining TLPs starting with the Sequence Number immediately after the Sequence Number in the Nak. If the Nak caused at least one TLP to be purged from the buffer, then we've made forward progress. In that case, the transmitter resets the REPLAY\_NUM counter and REPLAY\_TIMER and loads the AckD\_SEQ register with the Sequence Number of the Nak.

TLP Replay

When a Replay becomes necessary, the transmitter blocks acceptance of new TLPs from its Transaction Layer. It then replays the necessary TLPs in the buffer in the same order they were placed into the buffer (like a FIFO). After the replay event, the Data Link Layer unblocks acceptance of new TLPs from its Transaction Layer.

tion Layer. The replayed TLPs remain in the buffer until they are finally acknowledged at some later time.

## Efficient TLP Replay

Ack or Nak DLLPs received during replay must be processed. So there are two main options here, the transmitter may hold them until the replay is finished and then evaluate the Ack or Naks and take the appropriate steps. Another option would be to begin processing the new Ack/Nak DLLPs even during the replay. If this option is used, the newly received Ack might purge some entries from the buffer while replay is in progress, possibly reducing the number of TLPs that need to be replayed and saving time on the Link. This is allowed, but it is important to remember that once a TLP is started for transmission, it must be completed.

## Example of a Nak

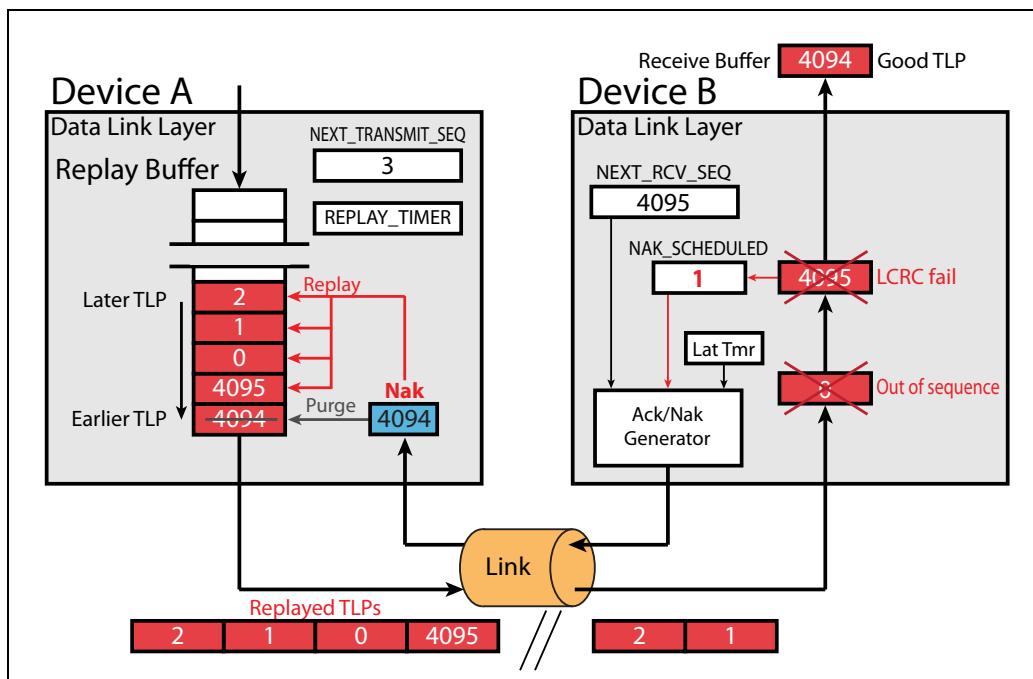
Consider Figure 10-10 on page 335.

1. Device A transmits TLPs with Sequence Number 4094, 4095, 0, 1, and 2.
2. Device B receives TLP 4094 without error and increments the NEXT\_RCV\_SEQ count to 4095 and starts the AckNak\_LATENCY\_TIMER.
3. Device B detects a CRC error in the next TLP received (TLP 4095) and sets the NAK\_SCHEDULED flag, which will cause a Nak to be sent with Sequence Number 4094 (NEXT\_RCV\_SEQ count - 1). Device B does NOT wait until the AckNak\_LATENCY\_TIMER expires before sending the Nak. It will typically be sent on the next packet boundary. In fact, since a Nak is scheduled for transmission, the AckNak\_LATENCY\_TIMER is stopped and reset.
4. Device B will continue evaluating incoming TLPs looking for TLP 4095. However, because Device A did not know there was a problem yet, it had sent packets 0, 1 and 2, which Device B will receive. However, Device B will not accept them, even though they may be good TLPs (meaning they did not fail the LCRC check). This is because all packets have to be accepted in order. So Device B will simply drop those packets because they are considered out of sequence, but no additional Nak will be sent. Even if one or more of these TLPs fail the LCRC check, no additional NAK is sent. The NAK\_SCHEDULED flag is already set and it will only be cleared once Device B successfully receives the TLP it is expecting (TLP 4095 in this example).

# Chapter 10: Ack/Nak Protocol

5. Device A receives Nak 4094 and purges TLP 4094 and earlier TLPs (none in this example) from the Replay Buffer. Also, since forward progress was made, it resets the REPLAY\_TIMER and REPLAY\_NUM count.
6. Since the acknowledge DLLP received was a Nak and not an Ack, Device A then replays all remaining TLPs in the Replay Buffer (TLPs 4095, 0, 1, and 2) and restarts the REPLAY\_TIMER and increments the REPLAY\_NUM count by one.
7. Once Device B receives the replayed TLP 4095, it will clear the NAK\_SCHEDULED flag, increment the NEXT\_RCV\_SEQ count and start the AckNak\_LATENCY\_TIMER.

Figure 10-10: Example of a Nak



## Repeated Replay of TLPs

**General.** Each time the transmitter receives a Nak, it replays the buffer contents, and the 2-bit REPLAY\_NUM counter is incremented to keep track of the number of replay events. The replay caused by a Nak in the previous example will increment REPLAY\_NUM.

If the replay doesn't clear the problem, though, we enter a new situation. The receiver has set the Nak Scheduled Flag and cannot send any more Ack or Naks until it sees the offending TLP correctly received. If the replay doesn't make that happen for some reason, then there will be no response from the receiver. What saves us now is the transmitter's REPLAY\_TIMER. When it times out, the entire contents of the Replay Buffer will be resent, the REPLAY\_NUM counter will be incremented and the REPLAY\_TIMER will be reset and restarted. If the REPLAY\_TIMER expires without receiving an Ack or Nak indicating forward progress, this replay process can be repeated up to three times. If after the third replay, there is still no forward progress and the REPLAY\_TIMER expires again, this would cause the REPLAY\_NUM counter to roll over from 3 back to 0.

**Replay Number Rollover.** When this happens, the assumption is that there must be something wrong with the Link, so the Link Layer triggers the Physical Layer to re-train the Link, causing it to go into the Recovery State (see "Recovery State" on page 571). If the optional Advanced Error Reporting registers are implemented, the Replay Number Rollover error status bit will also be set ("Advanced Correctable Error Handling" on page 688). The Replay Buffer contents are preserved and the Link Layer is not initialized during the re-training process (this is simply re-training the Link, not performing a reset of the Link). When re-training completes, the transmitter resumes the same replay process again in hopes that the problem has been cleared and the TLPs can now be replayed successfully.

The spec does not describe how a device might handle repeated rollover events if the Link training doesn't clear the problem. The author has seen commercially available hardware that had no mechanism to detect this condition and got stuck in an endless loop of re-training. It seems good therefore, to recommend that a device track the number of re-train attempts. After sufficient attempts, the device could signal an Uncorrectable Fatal Error or an interrupt as a way to notify software of this condition.

## Replay Timer

The transmitter REPLAY\_TIMER is running anytime there are TLPs that have been transmitted but have not yet been acknowledged. The goal of the REPLAY\_TIMER is to ensure that TLPs are being acknowledged in a timely fashion. If this timer expires, it indicates that an Ack or Nak should have been received by that point in time, so something must have gone wrong and the fix from the transmitter's point-of-view is to perform a replay, meaning to re-send everything in the Replay Buffer.

# Chapter 10: Ack/Nak Protocol

---

Based on the purpose of this timer, it makes sense that its timeout value should be correlated the AckNak\_LATENCY\_TIMER in the receiver. In fact, the REPLAY\_TIMER is simply three times longer than the AckNak\_LATENCY\_TIMER.

A formula in the spec determines the timer's count value. Its expiration triggers a replay event and increments the REPLAY\_NUM counter. A couple of cases where timeout may arise is if an Ack or Nak is lost en route, or because an error in the receiver prevents it from returning an Ack or Nak. Timer-related rules:

- If not already running, the timer starts when the last symbol of any TLP is transmitted
- The timer is reset and restarted when:
  - An Ack indicating forward progress is received, AND there are still unacknowledged TLPs in the Replay Buffer
  - A Replay event occurs and the last symbol of the first replayed TLP is sent
- The timer is reset and held when:
  - There are no TLPs to transmit, or the Replay Buffer is empty
  - A Nak is received; it restarts when the last symbol of the first replayed TLP is sent
  - The timer expires; it restarts when the last symbol of the first replayed TLP is sent
  - The Data Link Layer is inactive
- The timer is held during Link training or re-training

**REPLAY\_TIMER Equation.** The timeout value depends primarily on the max data payload and the width of the Link. The equation to calculate the REPLAY\_TIMER value in symbol times is given below. Note that the value is simply three times the Ack/Nak Latency value.

$$\left( \frac{(\text{Max\_Payload\_Size} + \text{TLPOverhead}) * \text{AckFactor}}{\text{LinkWidth}} + \text{InternalDelay} \right) * 3 + \text{Rx\_LoS\_Adjustment}$$

*(this term removed  
for Gen2 and later)*

The equation fields are defined as follows:

- **Max\_Payload\_Size** - the value in the Device Control Register. In the case of multiple Functions with different Max\_Payload\_Size values, the spec recommends using the smallest one of them.

- **TLP Overhead** - the additional TLP fields beyond the data payload (sequence number, header, digest, LCRC and Start/End framing symbols). In the spec, the overhead value is treated as a constant of 28 symbols.
- **AckFactor (AF)** - is basically a fudge factor representing the number of max payload-sized TLPs that can be received before an Ack must be sent. The AF value ranges from 1.0 to 3.0 and is intended to balance Link bandwidth efficiency and Replay Buffer size. The table in Figure 10-11 on page 339 shows the Ack Factor values for various link widths and payload sizes. These Ack Factor values are chosen to allow implementations to achieve good performance without requiring a large uneconomical buffer.
- **LinkWidth** - ranges from x1 (1-bit wide) to x32 (32-bits wide).
- **InternalDelay** - the internal delay of processing a TLP within the receiver and DLLPs (Acks) within the transmitter. This value is defined in the spec in symbol times, and depends on the Link speed: Gen1 = 19, Gen2 = 70, Gen3 = 115.
- **Rx\_L0s\_Adjustment** - This is a value that was included in the 1.x PCIe specs but was dropped for 2.0 and later PCIe specs. It could be used to account for the time required by the receive circuits to exit from L0s to L0. Setting the Extended Sync bit of the Link Control register affects the exit time from L0s and must be taken into account in this adjustment. Interestingly, the spec writers chose to assume this to be zero when creating their table of Replay Timer values. More on this in the following section.

**REPLAY\_TIMER Summary Table.** Figure 10-11 on page 339 is a summary table for the Gen1 rate that shows timer load values for various values of the variables in the REPLAY\_TIMER equation. The numbers have changed for the newer generations of the spec, and the new tables and a discussion of them can be found in the section called “Timing Differences for Newer Spec Versions” on page 350. The tolerance for all of the table values is -0% to +100%.

Note that the table values in the spec (copied here for convenience) are considered “unadjusted” because they leave out the last item of the equation involving the time to recover from L0s. No explanation is given for this in the spec, but if the Link had to wake up from L0s to L0 just to replay a packet in case the timeout might have been an error, that would be poor power management.

## Chapter 10: Ack/Nak Protocol

A simple way to avoid this problem altogether is for the transmitter to ensure that the Replay Buffer is empty before entering L0s. The spec requires that step for entry into L1 but not L0s, and the reason probably has to do with the relative risk involved. Going to L1 requires a longer recovery process back to L0 that has some small risk of failure. If it fails to recover, the Physical Layer state machine will have to do more of the Link training, a process that clears the LinkUp flag to the Link Layer, causing the Link Layer to re-initialize. If there were entries in the Replay Buffer when that happened they'd be lost and problems could result. The recovery risk from L0s was evidently considered low enough not to warrant that requirement. Still, the L0s latency was left out when the table was constructed, leaving the reader to wonder about this. In the author's opinion, the spec writers expected designers to take steps to ensure that a Replay Timer timeout either doesn't occur while in L0s (by emptying the Replay Buffer before L0s entry), or will be delayed if the path for the Acks is observed to be in L0s.

Figure 10-11: Gen1 Unadjusted REPLAY\_TIMER Values

Max_Payload Size	X1 Link	X2 Link	X4 Link	X8 Link	X12 Link	x16 Link	X32 Link
128 Bytes	711	384	219	201	174	144	99
256 Bytes	1248	651	354	321	270	216	135
512 Bytes	1677	867	462	258	327	258	156
1024 Bytes	3213	1635	846	450	582	450	252
2048 Bytes	6285	3171	1614	834	1095	834	444
4096 Bytes	12,429	6243	3150	1602	2118	1602	828

The table summarizes values calculated using the equation, minus the Rx\_L0s\_Adjustment term

Example: Assume a 2-lane link with a Max\_Payload of 2048 bytes.

$$\left[ \frac{(\text{Max_Payload_Size} + \text{TLP Overhead}) * \text{AckFactor} + \text{Internal Delay}}{\text{LinkWidth}} \right] * 3$$

$$\left[ \frac{(2048 + 28) * 1.0 + 19}{2} \right] * 3 = 3171 \text{ (about a 12.7uS timeout period)}$$

## Transmitter DLLP Handling

The Ack/Nak Error Checking block determines whether there is an error in the 16-bit CRC of a received DLLP. If an error is detected, the DLLP is discarded. This is considered a correctable error and may have been set up to be reported in the optional Advanced Error Reporting registers (see Bad DLLP in “Advanced Correctable Error Handling” on page 688), but no further action is taken because this isn’t really a problem. The next successfully received DLLP of that type will bring the counters back up to speed. Consequently, TLPs might be purged a little later than they would have been or a replay may happen at a later time, but no information is lost. Of course, if the delay between successful Ack becomes too large, the REPLAY\_TIMER could expire, causing the TLPs to be replayed.

---

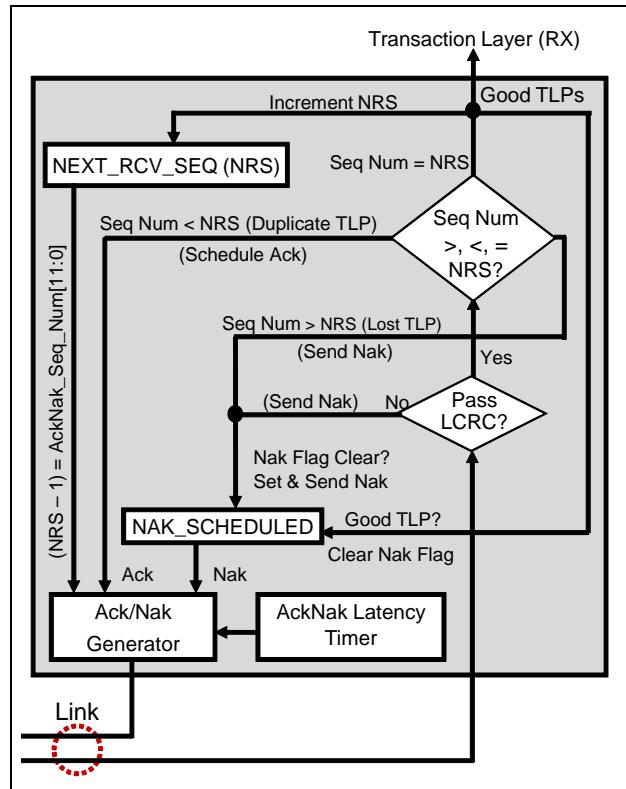
## Receiver Protocol Details

### Physical Layer

TLPs received at the Physical Layer are checked for receiver errors (such as framing, disparity, and invalid symbols). If there are errors at this level, the TLP is discarded and the Link Layer may be informed by some design-specific method so it can schedule a Nak and have the packet replayed. If the Link Layer is not informed, then eventually it will detect a Sequence Number violation and that will cause a Nak and a replay.

# Chapter 10: Ack/Nak Protocol

Figure 10-12: Ack/Nak Receiver Elements



## TLP LCRC Check

If there were no Physical Layer errors, the Link Layer checks first for CRC errors. The receiver calculates an expected LCRC value from the received TLP (excluding the LCRC field) and compares this value with the TLP's 32-bit LCRC. If the two match, the TLP is good. Otherwise, the TLP is discarded and the receiver schedules a Nak.

## Next Received TLP's Sequence Number

If the LCRC was correct, the receiver next compares the NEXT\_RCV\_SEQ counter against the Sequence Number that should be in the newly-received TLP. Under normal operational conditions, these two numbers will match. If they do, the receiver forwards the TLP to the Transaction Layer, increments the NEXT\_RCV\_SEQ counter, and schedules an Ack.

# PCI Express Technology

---

If the received TLP's Sequence Number turns out to be earlier or later than the NEXT\_RCV\_SEQ count, we have one of two cases: a duplicate TLP or an out of sequence TLP.

**Duplicate TLP.** If the Sequence Number of the incoming packet is earlier (logically smaller) than the expected value, it means the transmitter decided to resend a packet that the receiver has already seen before. This duplicate packet is not an error although we are wasting time on the Link by resending it. This might be caused by a timeout at the transmitter if the Ack or Nak for a previous TLP failed. When this is seen at the receiver, the duplicate packet is discarded and an Ack is scheduled with the Sequence Number of the last good TLP it has received (which is probably not the same Sequence Number in the replayed TLP).

**Out of Sequence TLP.** If the Sequence Number of the incoming packet is later (logically larger) than the expected value, the only explanation is that a TLP must have been lost. This is a correctable error and is handled by sending a Nak. It doesn't matter if the incoming packet is good because they can only be accepted in correct Sequence Number order. The packet is discarded and the receiver waits for a TLP with the expected Sequence Number.

The NEXT\_RCV\_SEQ counter is not incremented when a TLP is received with a CRC error, or was nullified, or for which the Sequence Number check fails.

A transmitter orders TLPs according to the PCI ordering rules to maintain correct program flow and avoid potential deadlock and livelock conditions (see Chapter 8, entitled "Transaction Ordering," on page 285). The Receiver is required to preserve this order and applies these three rules:

- When the receiver detects a bad TLP, it discards the TLP and all new TLPs that follow in the pipeline until the replayed TLPs are detected.
- Duplicate TLPs are discarded.
- TLPs received while waiting for a lost or corrupt TLP are discarded.

## Receiver Schedules An Ack DLLP

If the Data Link Layer of the receiver does not detect an error in an incoming TLP, it forwards the TLP to the Transaction Layer. The NEXT\_RCV\_SEQ counter is incremented and the receiver starts the AckNak\_LATENCY\_TIMER (assuming it was not already running). This is the equivalent of "scheduling an Ack." The receiver is allowed to continue receiving good TLPs without sending an Ack until the AckNak\_LATENCY\_TIMER expires. When the timer expires it

# Chapter 10: Ack/Nak Protocol

---

sends just one Ack with the Sequence Number of the last good TLP, acknowledging good receipt of all received TLPs up to the Sequence Number in the current Ack. This technique improves Link efficiency by reducing Ack/Nak traffic. For review, recall that this technique works because the TLPs must always be successfully received in order.

## Receiver Schedules a Nak

As mentioned earlier in the discussion of the receiver logic (see “Receiver Elements” on page 324), when the receiver detects an error on a TLP, it discards the bad packet and sets the NAK\_SCHEDULED flag if it was clear, which will cause a Nak to be scheduled with the Sequence Number of NEXT\_RCV\_SEQ count - 1. Since a Nak is now scheduled, the AckNak\_LATENCY\_TIMER is reset and halted. Scheduling a Nak can be thought of as being an “edge-triggered” event instead of a level-triggered event. It is seeing the rising edge of the NAK\_SCHEDULED flag that causes a Nak to be scheduled. Another Nak cannot be sent until the next rising edge, which means the NAK\_SCHEDULED flag must be cleared (falling edge) first. There are only two events that will clear the NAK\_SCHEDULED flag. The first is successfully receiving the expected next TLP (TLP with a Sequence Number that matches the NEXT\_RCV\_SEQ count). The second is a reset of the link (not retraining, but reset).

Although it’s important to get the Nak to the transmitter quickly (no other TLPs can be accepted until the failed one is seen without errors), other outgoing TLPs, DLLPs or Ordered Sets already be in progress or have a higher priority than the Nak which means the receiver would have to delay the transmission of the Nak until they’re done (see “Recommended Priority To Schedule Packets” on page 350). In the meantime, if other TLPs arrive at the receiver they are discarded and no additional Acks or Naks will be scheduled while the NAK\_SCHEDULED flag is set.

## AckNak\_LATENCY\_TIMER

This timer defines how long a receiver can wait before it must send an Ack for a successfully received TLP (or sequence of TLPs). As stated before, this timer is running anytime a receiver successfully receives a TLP that it has not yet acknowledged. Once the timer expires, an Ack is scheduled for transmission with the Sequence Number of the last good TLP it received. Scheduling an Ack resets the AckNak\_LATENCY\_TIMER and it only starts counting again once the next TLP is successfully received.

# PCI Express Technology

---

## AckNak\_LATENCY\_TIMER Equation.

The timeout value for the AckNak\_LATENCY\_TIMER is defined by the spec and varies based on the Negotiated Link Width and Max Payload Size Enabled. The equation which defines the timeout is shown below:

$$\frac{(\text{Max\_Payload\_Size} + \text{TLPOverhead}) * \text{AckFactor}}{\text{LinkWidth}} + \text{InternalDelay} + \text{Tx\_L0s\_Adjustment}$$

*(this term removed  
for Gen2 and later)*

The value in the timer is given in symbol times, the time it takes to send one symbol across the Link: 4ns for Gen1, 2ns for Gen2, and 1ns for Gen3.

The equation fields are:

- **Max\_Payload\_Size** - the value in the Device Control Register. In the case of multiple Functions with different Max\_Payload\_Size values, the spec recommends using the smallest one of them.
- **TLPOverhead** - the additional TLP fields beyond the data payload (sequence number, header, digest, LCRC and Start/End framing symbols). In the spec, the overhead value is treated as a constant of 28 symbols.
- **AckFactor** (AF) - is basically a fudge factor representing the number of max payload-sized TLPs that can be received before an Ack must be sent. The AF value ranges from 1.0 to 3.0 and is intended to balance Link bandwidth efficiency and Replay Buffer size. The table in Figure 10-11 on page 339 shows the Ack Factor values for various link widths and payload sizes. These Ack Factor values are chosen to allow implementations to achieve good performance without requiring a large uneconomical buffer.
- **LinkWidth** - ranges from x1 (1-bit wide) to x32 (32-bits wide).- from 1 to 32 Lanes.
- **InternalDelay** - the internal delay of processing a TLP within the receiver and DLLPs (Acks) within the transmitter. This value is defined in the spec in symbol times, and depends on the Link speed: Gen1 = 19, Gen2 = 70, Gen3 = 115.
- **Tx\_L0s\_Adjustment**: - This is a value that was included in the 1.x PCIe specs but was dropped for 2.0 and later PCIe specs. It could be used to account for the time required by the receive circuits to exit from L0s to L0. Setting the Extended Sync bit of the Link Control register affects the exit time from L0s and must be taken into account in this adjustment. Interestingly, the spec writers chose to assume this to be zero when creating their table of Replay Timer values.

# Chapter 10: Ack/Nak Protocol

---

**AckNak\_LATENCY\_TIMER Summary Table.** Figure 10-2 on page 345 shows the Gen1 timer load values for all the possible values used in the AckNak\_LATENCY\_TIMER equation. Higher data rates change the equation and the resulting table (see “Timing Differences for Newer Spec Versions” on page 350). Like the Replay Timer table, this table is constructed by assuming the L0s adjustment in the equation is zero and then referring to the resulting values as ‘unadjusted’. Note that the tolerance for all of the table values is -0% to +100%.

Table 10-2: Gen1 Unadjusted Ack Transmission Latency

Max_Payload Size	X1 Link	X2 Link	X4 Link	X8 Link	X12 Link	x16 Link	X32 Link
128 Bytes	237 (AF=1.4)	128 (AF=1.4)	73 (AF=1.4)	67 (AF=2.5)	58 (AF=3.0)	48 (AF=3.0)	33 (AF=3.0)
256 Bytes	416 (AF=1.4)	217 (AF=1.4)	118 (AF=1.4)	107 (AF=2.5)	90 (AF=3.0)	72 (AF=3.0)	45 (AF=3.0)
512 Bytes	559 (AF=1.0)	289 (AF=1.0)	154 (AF=1.0)	86 (AF=1.0)	109 (AF=2.0)	86 (AF=2.0)	52 (AF=2.0)
1024 Bytes	1071 (AF=1.0)	545 (AF=1.0)	282 (AF=1.0)	150 (AF=1.0)	194 (AF=2.0)	150 (AF=2.0)	84 (AF=2.0)
2048 Bytes	2095 (AF=1.0)	1057 (AF=1.0)	538 (AF=1.0)	278 (AF=1.0)	365 (AF=2.0)	278 (AF=2.0)	148 (AF=2.0)
4096 Bytes	4143 (AF=1.0)	2081 (AF=1.0)	1050 (AF=1.0)	534 (AF=1.0)	706 (AF=2.0)	534 (AF=2.0)	276 (AF=2.0)

## More Examples

In the classroom setting examples often make it much easier to grasp the Ack/Nak process and so some of them are presented here to illustrate special cases.

---

## Lost TLPs

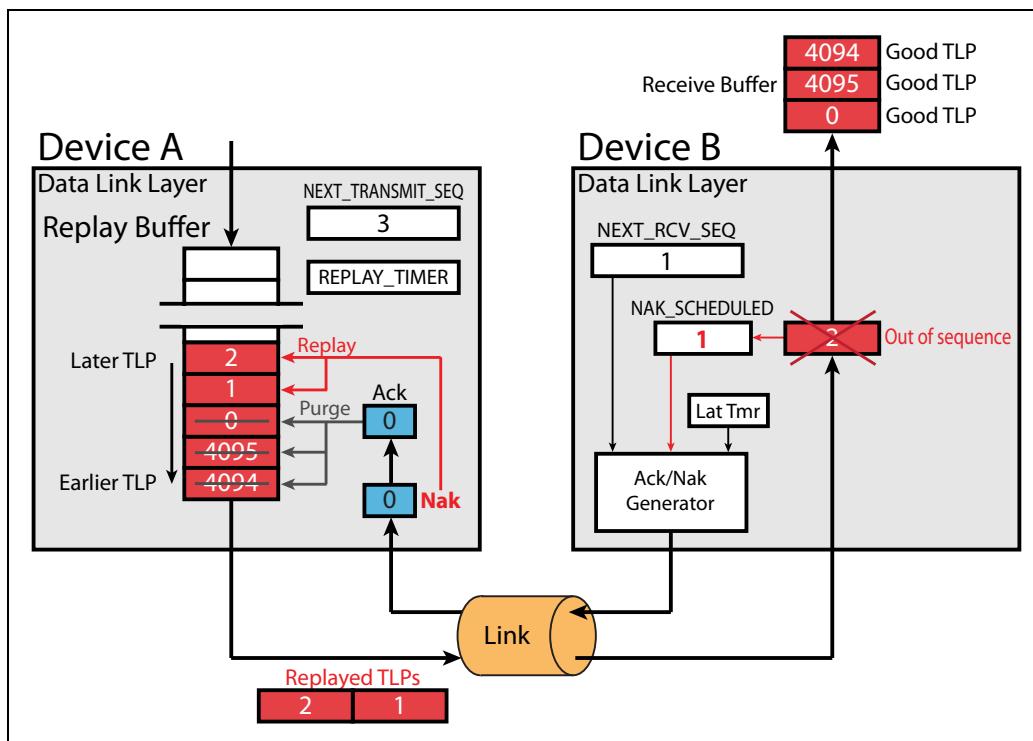
Consider Figure 10-13 on page 346, showing how a lost TLP is detected and handled.

1. Device A transmits TLPs 4094, 4095, 0, 1, and 2.
2. Device B successfully receives TLP 4094 so it starts its AckNak\_LATENCY\_TIMER and increments its NEXT\_RCV\_SEQ count. After that, it also receives TLPs 4095 and 0.

# PCI Express Technology

3. After receiving TLP 0, the AckNak\_LATENCY\_TIMER expires which causes it to schedule an Ack with Sequence Number of 0.
4. Seeing Ack 0, Device A purges TLPs 4094, 4095, and 0 from its replay buffer.
5. TLP 1 is lost en route for some reason (maybe the Physical Layer dropped it), and TLP 2 arrives instead. The Sequence Number check shows Device B that TLP 2's Sequence Number is not equal to the NEXT\_RCV\_SEQ count but is in the out of sequence range.
6. Device B discards TLP 2 and sets the NAK\_SCHEDULED flag which will send a Nak 0 (NEXT\_RCV\_SEQ count - 1) in this case.
7. Upon receipt of Nak 0, Device A replays TLPs 1 and 2. It would purge TLP 0 and any earlier ones in the Replay Buffer, but they were removed earlier so that becomes unnecessary.
8. TLPs 1 and 2 arrive without error at Device B and are forwarded to the Transaction Layer.

Figure 10-13: Handling Lost TLPs



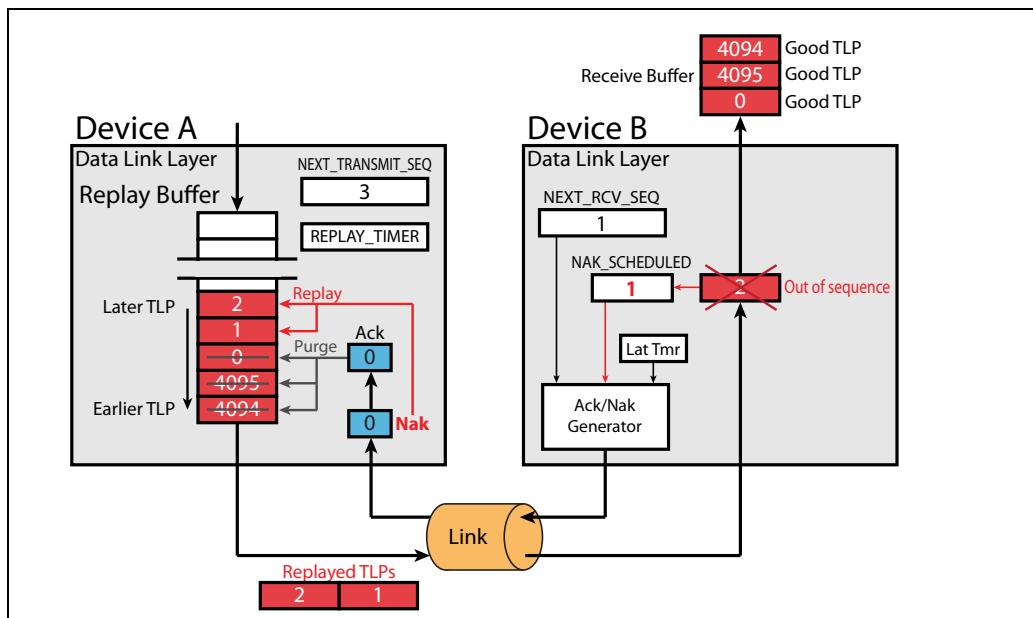
## Bad Ack

Figure 10-14 on page 347 which shows the protocol for handling a corrupt Ack.

1. Device A transmits TLPs 4094, 4095, 0, 1, and 2.
2. Device B receives TLPs 4094, 4095, and 0, sets NEXT\_RCV\_SEQ to 1, and returns Ack 0 because the AckNak\_LATENCY\_TIMER had expired.
3. Ack 0 has a bit during its flight on the Link, so when Device A checks its 16-bit CRC, it fails the check and is discarded. This means TLPs 4094, 4095, and 0 remain in Device A's Replay Buffer.
4. TLPs 1 and 2 arrive at Device B and are good, so NEXT\_RCV\_SEQ count increments to 3 and Ack 2 is returned once the AckNak\_LATENCY\_TIMER expires again.
5. Ack 2 arrives safely at Device A, which purges its Replay Buffer of TLPs 4094, 4095, 0, 1, and 2.

If Ack 2 is also lost or corrupted and no further Ack or Nak DLLPs are returned to Device A, its REPLAY\_TIMER expires causing a replay of its entire buffer. Device B sees TLPs 4094, 4095, 0, 1 and 2 and considers them to be duplicates [their sequence numbers are earlier than NEXT\_RCV\_SEQ count (3)]. They are discarded and *another* Ack 2 would be returned to Device A because of the duplicate packets.

Figure 10-14: Handling Bad Ack



---

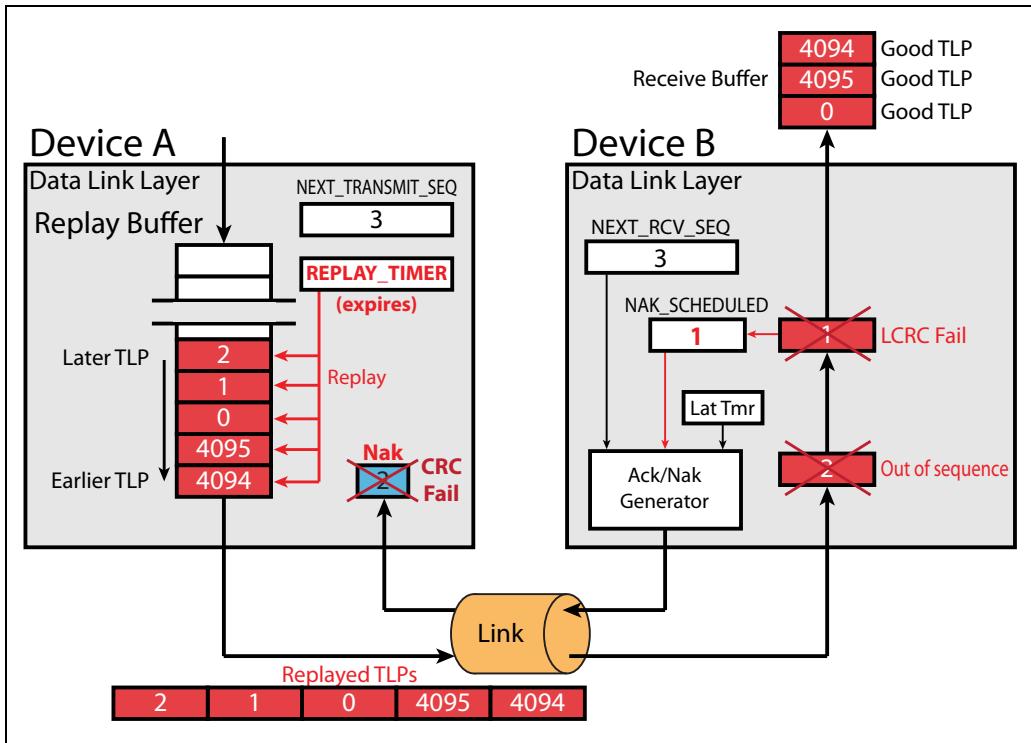
## Bad Nak

Figure 10-15 on page 349 which shows protocol for handling a corrupt Nak.

1. Device A transmits TLPs 4094, 4095, 0, 1, and 2.
2. Device B receives TLPs 4094, 4095, and 0 all successfully (and the AckNak\_LATENCY\_TIMER has not yet expired). The next TLP that it receives fails the LCRC check, so Device B sets the NAK\_SCHEDULED flag, and resets and holds the AckNak\_LATENCY\_TIMER. The Nak is transmitted back with a Sequence Number of the last good TLP received, 0.
3. Nak 0 fails the 16-bit CRC check at Device A and is discarded.
4. At this point, Device B will not be sending anymore Ack or Naks until it successfully receives the next TLP it is expecting, TLP 1 in this example. However, this will require a replay. Device A does not yet know that a replay is required because the one Nak that was sent back was corrupted and discarded. This gets resolved by the REPLAY\_TIMER. The REPLAY\_TIMER will eventually expire because it has not seen an Ack or Nak that makes forward progress in the specified time frame.
5. Once the REPLAY\_TIMER expires, Device A will replay all TLPs in the Replay Buffer, increment REPLAY\_NUM count and reset and restart the REPLAY\_TIMER.
6. Device B will receive TLPs 4094, 4095 and 0 and recognize that they are duplicates. The duplicate TLPs will be dropped and an Ack will be scheduled with a Sequence Number 0 (indicating the furthest progress made).
7. Once TLP 1 is successfully received by Device B, it will clear the NAK\_SCHEDULED flag, increment the NEXT\_RCV\_SEQ and restart the AckNak\_LATENCY\_TIMER because it has successfully received a TLP that it has not yet acknowledged.

# Chapter 10: Ack/Nak Protocol

Figure 10-15: Handling Bad Nak



## Error Situations Handled by Ack/Nak

The Ack/Nak protocol guarantees reliable delivery of TLPs despite several possible errors. The list of errors below includes the correction mechanism used to resolve them.

- **LCRC error in a TLP.** **Solution:** Receiver detects LCRC error and schedules a Nak that contains the NEXT\_RCV\_SEQ count - 1. In response, the transmitter replays at least one TLP, starting with the one that failed.
- **TLPs lost en route to the receiver's Data Link Layer (e.g. Physical Layer detects issue with packet and drops it).** **Solution:** The receiver checks the Sequence Number on all received TLPs, expecting them to arrive with the next sequential Sequence Number. If a TLP is lost, the Sequence Number of the next one that succeeds will be out of sequence. In response, the Receiver

# PCI Express Technology

---

schedules a Nak with NRS count - 1, and the transmitter replays at least one TLP, starting with the missing one.

- **Corrupted Ack or Nak en route to the transmitter.** **Solution:** The Transmitter detects a CRC error in the DLLP (see “Receiver handling of DLLPs” on page 309), discards the packet and simply waits for the next one.
  - **Ack Case:** A subsequent Ack received with a later Sequence Number causes the transmitter Replay Buffer to purge all TLPs with Sequence Numbers equal to or earlier than it. The transmitter is unaware that anything was wrong (except for a potential case of the Replay Buffer temporarily filling up).
  - **Nak Case:** The receiver, having set the Nak Scheduled flag, will not send another Nak or any Ack until it successfully receives the next expected TLP, meaning a replay is needed. Of course, the transmitter doesn’t know it needs to replay if the Nak was lost. In this case, the REPLAY\_TIMER will eventually expire and trigger the replay.
- **No Ack/Nak seen** within the expected time. **Solution:** REPLAY\_TIMER timeout triggers a replay.
- **Receiver fails to send Ack/Nak** for a received TLP. **Solution:** Again, the transmitter’s REPLAY\_TIMER will expire and result in a replay.

---

## Recommended Priority To Schedule Packets

A device may have many types of TLPs, DLLPs and Ordered Sets to transmit on a given Link. The recommended priority for scheduling packets is:

1. Completion of any TLP or DLLP currently in progress (highest priority)
2. Ordered Set
3. Nak
4. Ack
5. Flow Control
6. Replay Buffer re-transmissions
7. TLPs that are waiting in the Transaction Layer
8. All other DLLP transmissions (lowest priority)

---

## Timing Differences for Newer Spec Versions

As mentioned earlier, the timer values for the Ack/Nak protocol are different for Gen2 and later versions of the spec. To improve readability of the text, only the Gen1 versions (2.5 GT/s rate) were included in the earlier discussion, but all three versions are included here for convenience.

# Chapter 10: Ack/Nak Protocol

---

As before, the values given are in symbol times, so the actual time is that value multiplied by the time needed to deliver one symbol over the Link at that rate. For review, the time to transmit one symbol (known as a Symbol Time) is 4ns for Gen1, 2ns for Gen2, and 1.25ns to transmit 1 byte for Gen3.

---

## Ack Transmission Latency (AckNak Latency)

One interesting difference between the spec versions is the way the L0s recovery time is considered. In the 1.x specs, an argument is included in the AckNak\_LATENCY\_TIMER equation to account for this, but the tables in the spec based on that equation put its value at zero and call the resulting values ‘unadjusted’. Beginning with the 2.0 spec, the L0s recovery value is dropped from the equation altogether and the text states that the receiver is not required to adjust Ack scheduling based on L0s exit latency or the value of the Extended Sync bit. None of the table values contain an L0s recovery component and are therefore all still called ‘unadjusted’.

Note that, since the AF (Ack Factor) values are the same in all the tables and were shown in the earlier presentation of the Gen1 table, they’re not included in the tables here.

Also, as it was for Gen1, the tolerance for all of the table values is -0% to +100%. To illustrate this, Table 10-3 on page 351 lists the time for a x1 Link and Max Payload size of 128 Bytes as 237 symbol times. Legal values would therefore range from no less than 237 symbol times to no more than 474.

## 2.5 GT/s Operation

Table 10-3: Gen1 Unadjusted AckNak\_LATENCY\_TIMER Values (Symbol Times)

Max Payload	x1 Link	x2 Link	x4 Link	x8 Link	x12 Link	x16 Link	x32 Link
128 Bytes	237	128	73	67	58	48	33
256 Bytes	416	217	118	107	90	72	45
512 Bytes	559	289	154	86	109	86	52
1024 Bytes	1071	545	282	150	194	150	84
2048 Bytes	2095	1057	538	278	365	278	148

# PCI Express Technology

---

Table 10-3: Gen1 Unadjusted AckNak\_LATENCY\_TIMER Values (Symbol Times) (Continued)

Max Payload	x1 Link	x2 Link	x4 Link	x8 Link	x12 Link	x16 Link	x32 Link
4096 Bytes	4143	2081	1050	534	706	534	276

## 5.0 GT/s Operation

Table 10-4: Gen2 Unadjusted AckNak\_LATENCY\_TIMER Values (Symbol Times)

Max Payload	x1 Link	x2 Link	x4 Link	x8 Link	x12 Link	x16 Link	x32 Link
128 Bytes	288	179	124	118	109	99	84
256 Bytes	467	268	169	158	141	123	96
512 Bytes	610	340	205	137	160	137	103
1024 Bytes	1122	596	333	201	245	201	135
2048 Bytes	2146	1108	589	329	416	329	199
4096 Bytes	4194	2132	1101	585	757	585	327

## 8.0 GT/s Operation

Table 10-5: Gen3 Unadjusted AckNak\_LATENCY\_TIMER Values (Symbol Times)

Max Payload	x1 Link	x2 Link	x4 Link	x8 Link	x12 Link	x16 Link	x32 Link
128 Bytes	333	224	169	163	154	144	129
256 Bytes	512	313	214	203	186	168	141
512 Bytes	655	385	250	182	205	182	148
1024 Bytes	1167	641	378	246	290	246	180
2048 Bytes	2191	1153	634	374	461	374	244
4096 Bytes	4239	2177	1146	630	802	630	372

## Replay Timer

Much like the AckNak Latency Timer calculation, L0s recovery time is considered differently for the Replay Timer in newer spec versions. In the 1.x specs, an argument is included in the Replay Timer equation to account for this, but the tables in the spec based on that equation put its value at zero and call the resulting values ‘unadjusted’. Beginning with the 2.0 spec, the argument is dropped from the equation altogether and the text states that the transmitter should compensate for L0s exit if it will be used, either by statically adding that time to the table values or by sensing when the Link is in that state and allowing extra time in that case. The table values still don’t contain an L0s component and are still called ‘unadjusted’.

As a final word on this topic, the spec strongly recommends that a transmitter should not do a replay on a Replay Timer timeout if it’s possible that the delay in receiving an Ack was caused by the other device’s transmitter being in the L0s state.

Note that, just like for the Ack Latency Timer tables, the tolerance for all of the table values is -0% to +100%. To illustrate this, Table 10-6 on page 353 lists the time for a x1 Link and Max Payload size of 128 Bytes as 711 symbol times. Legal values would therefore range from no less than 711 symbol times to no more than 1422.

## 2.5 GT/s Operation

Table 10-6: Gen1 Unadjusted REPLAY\_TIMER Values in Symbol Times

Max Payload	x1 Link	x2 Link	x4 Link	x8 Link	x12 Link	x16 Link	x32 Link
128 Bytes	711	384	219	201	174	144	99
256 Bytes	1248	651	354	321	270	216	135
512 Bytes	1677	867	462	258	327	258	156
1024 Bytes	3213	1635	846	450	582	450	252
2048 Bytes	6285	3171	1614	834	1095	834	444
4096 Bytes	12429	6243	3150	1602	2118	1602	828

# PCI Express Technology

---

## 5.0 GT/s Operation

Table 10-7: Gen2 Unadjusted REPLAY\_TIMER Values in Symbol Times

Max Payload	x1 Link	x2 Link	x4 Link	x8 Link	x12 Link	x16 Link	x32 Link
128 Bytes	864	537	372	354	327	297	252
256 Bytes	1401	804	507	474	423	369	288
512 Bytes	1830	1020	615	411	480	411	309
1024 Bytes	3366	1788	999	603	735	603	405
2048 Bytes	6438	3324	1767	987	1248	987	597
4096 Bytes	12582	6396	3303	1755	2271	1755	981

## 8.0 GT/s Operation

Table 10-8: Gen3 Unadjusted REPLAY\_TIMER Values

Max Payload	x1 Link	x2 Link	x4 Link	x8 Link	x12 Link	x16 Link	x32 Link
128 Bytes	999	672	507	489	462	432	387
256 Bytes	1536	939	642	609	558	504	423
512 Bytes	1965	1155	750	546	615	546	444
1024 Bytes	3501	1923	1134	738	870	738	540
2048 Bytes	6573	3459	1902	1122	1383	1122	732
4096 Bytes	12717	6531	3438	1890	2406	1890	1116

---

## Switch Cut-Through Mode

Now that we've described how the protocol works, this is a good time to explain an exception to its general operation. PCIe supports a Switch feature, called 'cut-through mode', that can be used to improve the transfer latency for large TLPs through a Switch.

## Background

Consider an example where a large TLP needs to pass through a Switch as shown in Figure 10-16 on page 357. Since the Ingress Switch Port can't tell whether there was an error in the packet until it has seen the whole TLP, it'll normally store the entire packet and check it for errors before forwarding it to the Egress Port. This store-and-forward method works but, for large packets, the latency to get through the Switch can be large which may be an issue for some applications. It would be nice to minimize this latency if possible.

---

## A Latency Improvement Option

Since the first part of the TLP contains the header with the routing information for the packet, one option would be to assume that the packet is a good packet and start evaluating the routing info in header even before the entire packet is received. This would allow a Switch to begin forwarding the TLP to the Egress Port as soon as that routing is evaluated. The Egress Port could then go ahead and start sending it out onto its Link, as long as doing so will not cause an underflow condition within the Switch. (A potential underflow case could easily happen if the Ingress Port is x1 and the Egress Port is x16. The Egress Port would be sending the packet out much faster than it is being received.)

Of course, the Ingress Port can't check for errors in the packet until it receives the LCRC at the end of the packet, so there is a small risk involved that the TLP being forwarded out may actually contain an error. Eventually, the end of the TLP arrives at the Ingress Port and the packet can be checked. If it turns out there was an error, the Ingress Port takes the normal behavior to a bad TLP and simply sends a Nak to have the packet replayed. However, we now have to deal with the problem that most of a packet that we now know is bad has already been forwarded on to the Egress Port. What are our options at this point? We could finish forwarding the packet and wait for a Nak from the neighboring receiver when it sees the error, but the packet in the replay buffer would be the bad one, and so a replay there won't fix the problem. We might truncate the bad packet in flight, but the spec doesn't allow for that possibility. To make this work, we need another option, and that's where the Cut-Through option comes into play.

## Cut-Through Operation

Cut-through mode provides the solution to the forwarding problem described in the previous section: if an error is seen in the incoming packet, the packet that is already on its way out must be '**nullified**'.

A **nullified** packet is terminated with an EDB (end bad) symbol instead of an END (end good) symbol and, to make the condition very clear, the TLPs 32-bit LCRC is inverted (1's complement) from the original calculated value. In essence, a nullified packet is handled as though it had never existed. On the Switch Egress Port, that means the replay buffer discards the packet and the NEXT\_TRANSMIT\_SEQ counter is decremented by one (rolled back).

When a device receives a TLP that it recognizes as being a nullified TLP, it simply drops the packet and treats it as if it never existed. The NEXT\_RCV\_SEQ is not incremented, the AckNak\_LATENCY\_TIMER is not started, nor is the NAK\_SCHEDULED set. The receiving device silently discards the nullified TLP and does not return an Ack/Nak for it.

---

## Example of Cut-Through Operation

Figure 10-16 on page 357 illustrates a TLP coming in from the left, going through the Switch, and ending up at an Endpoint on the right. A TLP error occurs on the left Link. The steps are as follows:

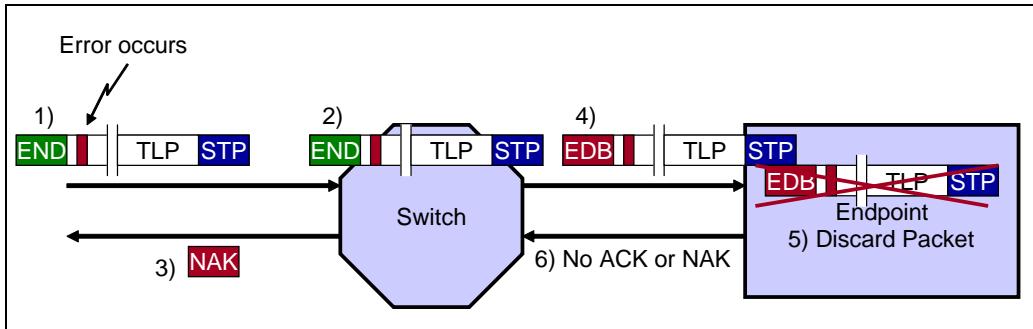
1. An incoming TLP is seen at the Switch Ingress Port. It has become corrupted in flight but that isn't known yet.
2. The TLP header arrives, is decoded, and the packet is forwarded to the destination Egress Port in cut-through operation.
3. Eventually, the end of the packet arrives and the Switch Ingress Port is able to complete the LCRC error check. It finds a CRC error and returns a Nak to the TLP source.
4. At the Egress Port, the Switch replaces the END framing symbol at the end of the bad TLP with EDB and inverts the calculated LCRC value. The TLP is now 'nullified' and the Switch discards it from the Replay Buffer.
5. The nullified packet arrives at the Endpoint. The Endpoint detects the EDB symbol and inverted LCRC and silently discards the packet. It does not return a Nak.

Now let's say the TLP source device replays the packet and no error occurs. As before, the TLP is forwarded to the Egress Port with very short latency. When

## Chapter 10: Ack/Nak Protocol

the rest of the TLP arrives at the Switch, there is no error, so an Ack is returned to the TLP source which then purges this TLP from its Replay Buffer. This time the Switch Egress Port keeps a copy of the TLP in its Replay Buffer. When the TLP reaches the destination, the packet has no errors and the Endpoint returns an Ack. Based on that, the Switch purges the copy of the TLP from its Replay Buffer and the sequence is complete.

Figure 10-16: Switch Cut-Through Mode Showing Error Handling



## **PCI Express Technology**

---

---

# Part Four:

# Physical Layer



---

# **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.

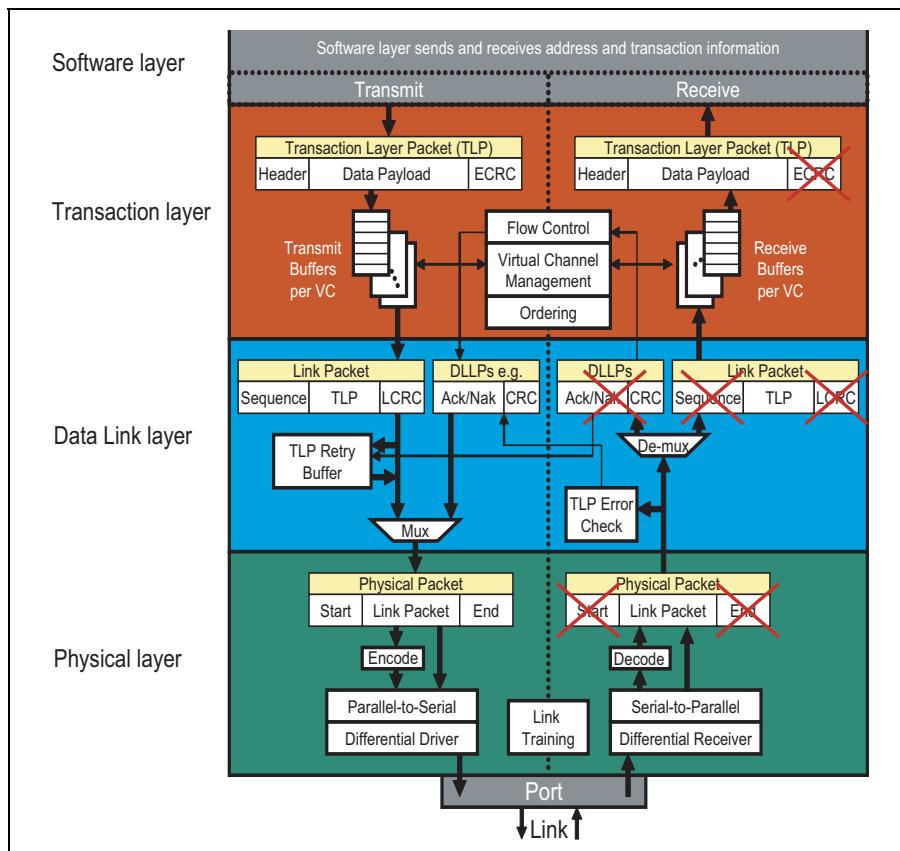
# PCI Express Technology

## 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



## Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

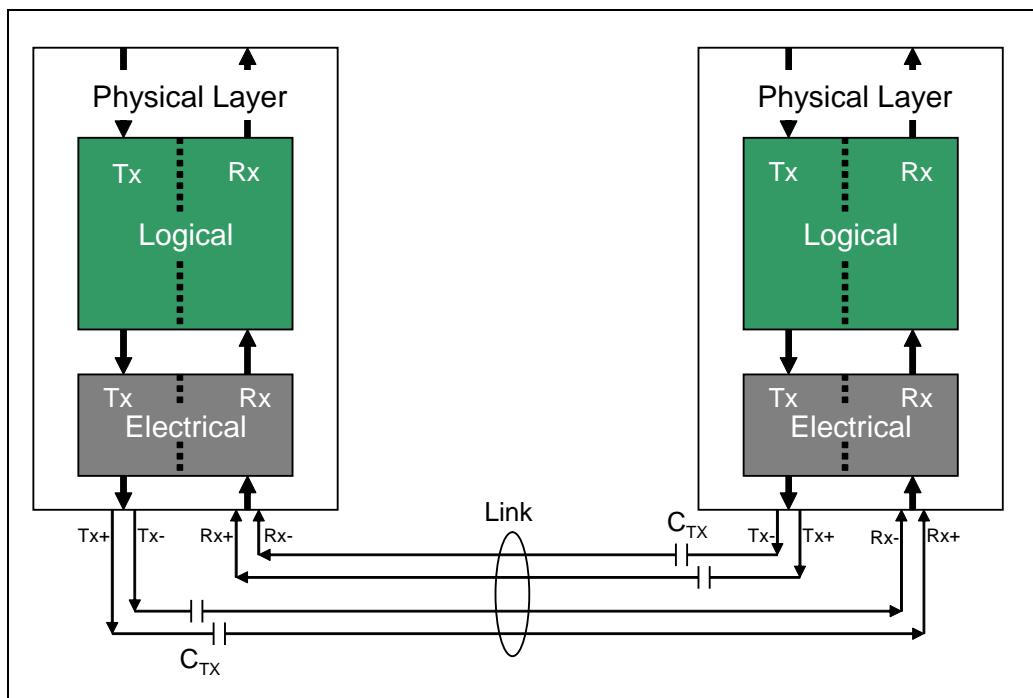
The contents of the layers are conceptual and don't define precise logic blocks, but to the extent that designers do partition them to match the spec their implementations can benefit because of the constantly increasing data rates affect the Physical Layer more than the others. Partitioning a design 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

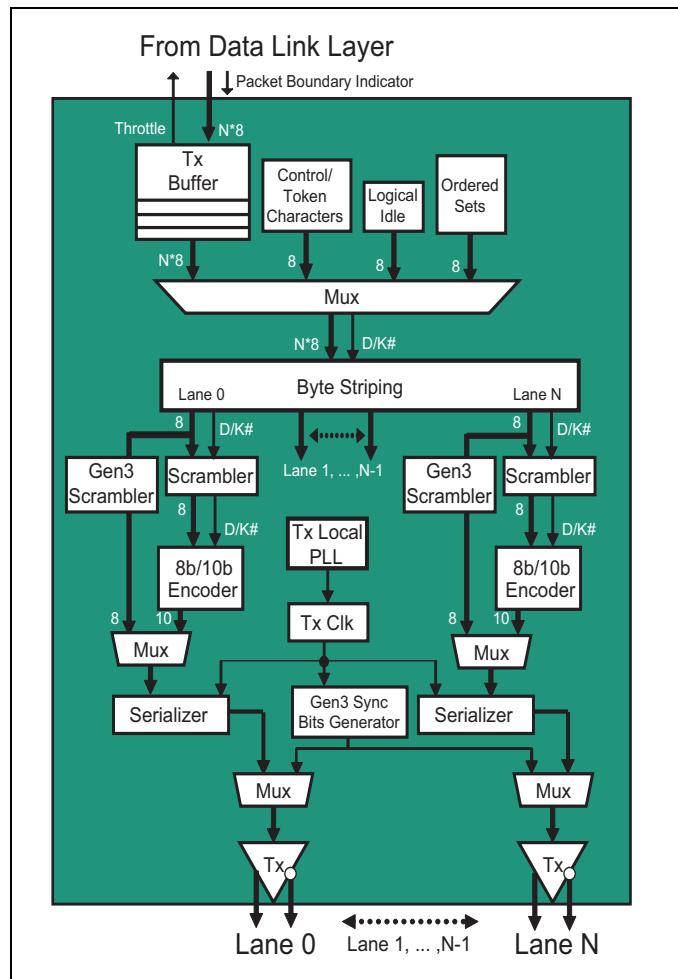
For simplicity, let's begin with a high-level overview of the transmit side of this layer, shown in Figure 11-3 on page 365. Starting at the top, we can see that packet bytes entering from the Data Link layer first go into a buffer. It makes sense to have a buffer here because there will be times when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

# Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

Figure 11-3: Physical Layer Transmit Details



Next, the parallel data bytes coming from the upper layers are sent to Byte Stripping logic where they are spread out, or striped, onto all the lanes of this link. One byte of the packet is transferred per lane, and all active lanes are used for each packet going out. The Lanes of the Link are all transmitting at the same time, so the bytes must come into this logic fast enough to accommodate that. For example, if there are eight Lanes, eight bytes of parallel from the upper layers may arrive at the byte-striping logic allowing data to be clocked onto all lanes simultaneously.

# PCI Express Technology

---

Next is the Scrambler, which XORs a pseudo-random pattern onto the outgoing data bytes to mix up the bits. Although it would seem that this might introduce problems, it doesn't because the scrambling pattern is predictable and not truly random, so the receiver can use the same algorithm to easily recover the original data. If the scramblers get out of step then the Receiver won't be able to make sense of the bit stream so, to guard against that problem, the scrambler is reset periodically (Gen1 and Gen2). That way, if the scramblers do get out of step with each other it won't be long before they're both re-initialized and back in step again. For Gen1 and Gen2 modes that re-initialization happens whenever the COM character is detected. For Gen3 mode, it happens whenever an EIEOS ordered set is seen. A more sophisticated 24-bit based scrambler is utilized in Gen3 mode, hence the alternate path through the Gen3 scrambler, as depicted in Figure 11-3 on page 365.

For Gen1 and Gen2 mode, the scrambled 8-bit characters are then encoded for transmission by the 8b/10b Encoder. Recall that a Character is an 8-bit un-encoded byte, while a Symbol is the 10-bit encoded output of the 8b/10b logic. There are several advantages to 8b/10b encoding, but it does add overhead.

For Gen3 a separate path is shown bypassing the encoder. In other words, scrambled bytes of a packet are transmitted without 8b/10b encoding. The Sync Bit Generator adds a 2-bit Sync Header prior to every 16 byte block of a packet. The added 2-bit Sync Header identifies the following 16 byte block to be either a data block or an ordered set block. This addition of a 2-bit Sync Header every 16 bytes (128 bits) is the basis of Gen3's 128b/130b encoding scheme.

Finally, the Symbols are serialized into a bit stream and forwarded to the electrical sub-block of the Physical Layer and transmitted to the other end of the link.

---

## Receive Logic Overview

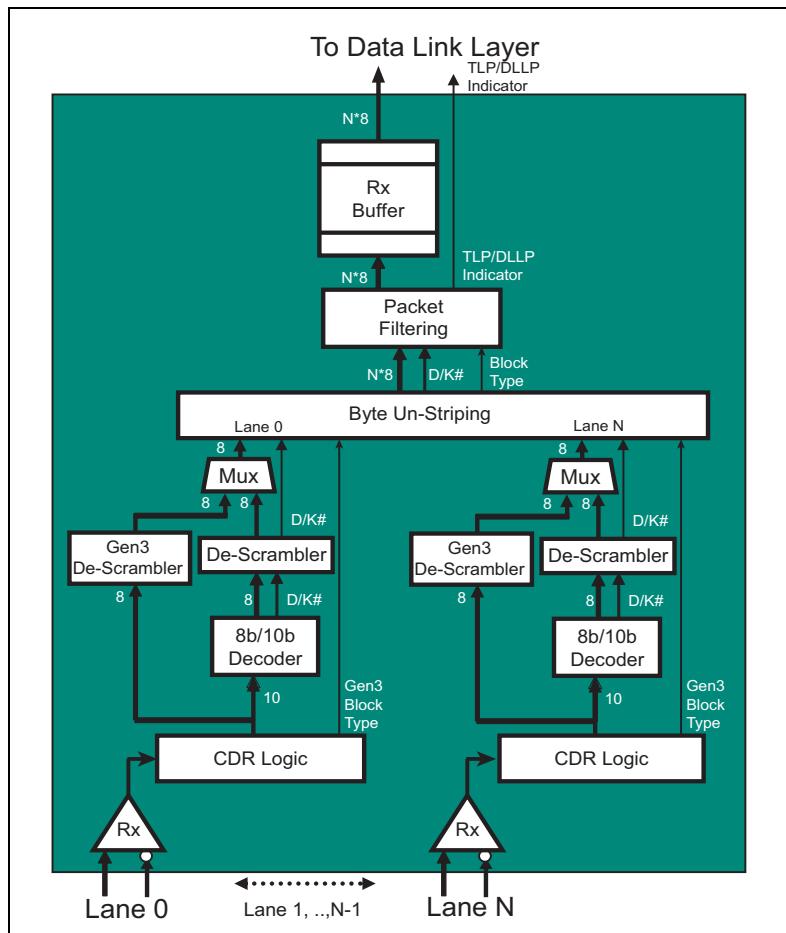
Figure 11-4 on page 367 shows the key elements that make up the receiver logic. The process described below is performed for each lane. Starting at the bottom this time, the first thing to mention is the receiver Clock and Data Recovery (CDR). The first step in this process is to recover the clock based on transitions in the incoming bit stream. This recovered clock faithfully reproduces the Transmitter's clock that was used to send the data and is used to latch the incoming bits into a deserializing buffer.

The next steps in the CDR process are to find the Gen1/Gen2 Symbol boundaries and divide the recovered clock by 10 to latch the 10-bit Symbols into the Elastic Buffer. For Gen3, the next step is to acquire Block Lock and then latch the 8-bit Symbols associated with each of the 16 bytes in the block into the Elastic Buffer — more on this in the next chapter.

## Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

Logic controlling the Elastic Buffer adjusts for minor clock variations between the recovered clock and the local clock of the receiver by adding or removing SKP Symbols as needed when an SOS (SKP Ordered Set) is detected. Finally, the Receiver's local clock moves each Symbol out of the Elastic Buffer.

*Figure 11-4: Physical Layer Receive Logic Details*



Using the 8b/10b Decoder, Gen1/Gen2 Symbols are decoded thus converting the 10-bit symbols to 8-bit characters. The descrambler applies the same scrambling method used at the transmitter to recover the original data. Finally, the bytes from each Lane are un-striped to form a byte stream that will be forwarded up to the Data Link Layer. Only TLPs and DLLPs are loaded into the receive buffer and sent to the Data Link Layer.

## Transmit Logic Details (Gen1 and Gen2 Only)

The section provides more detail associated with the steps identified in the previous section. Refer to the block diagram in Figure 11-5 on page 369 during this discussion.

---

### Tx Buffer

Starting from the top of the diagram once again, the buffer accepts TLPs and DLLPs from the Data Link Layer, along with ‘Control’ information that specifies when a new packet begins. As mentioned, the buffer allows us to stall the flow of characters from time to time in order to insert control characters and ordered sets. A ‘throttle’ signal is also shown going back up to the Data Link Layer to stop the flow of characters if the buffer should become full.

---

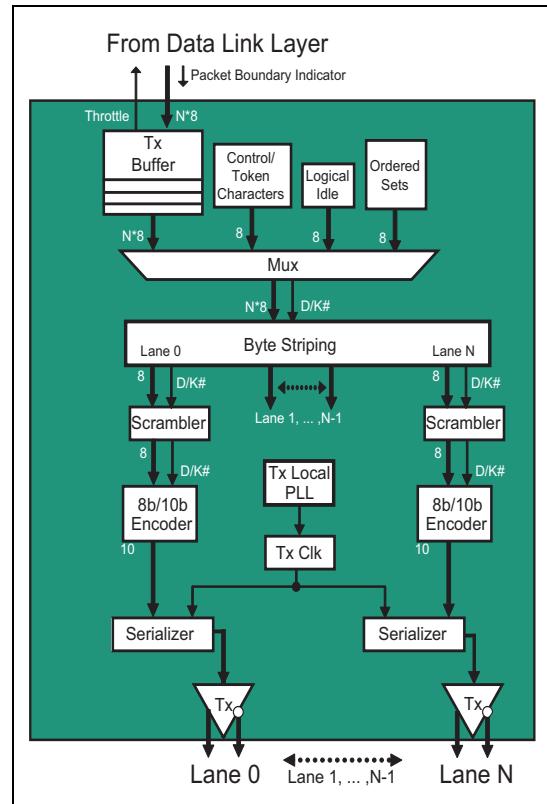
### Mux and Control Logic

The multiplexer, shown in Figure 11-6 on page 370, is used to insert special control (K) characters into the data flow coming from the buffer. Only the Physical Layer uses K control characters; they are inserted during transmission and removed at the receiver. The four different inputs to the mux are:

- **Transmit Data Buffer.** When the Data Link Layer supplies a packet, the mux gates the character stream through. All of the characters coming from the buffer are D characters, so the D/K# signal is driven high when Tx Buffer contents are gated.
- **Start and End characters.** These Control characters are added to the start and end of every TLP and DLLP (see Figure 11-7 on page 371) and allow a receiver to readily detect the boundaries of a packet. There are two Start characters: STP indicates the start of a TLP, while SDP indicates the start of a DLLP. An indicator from the Data Link Layer, along with the packet type, determines what type of framing character to insert. There are also two end characters, the End Good character (END) for normal transmission, and the End Bad character (EDB) to handle some error cases. Start and End characters are K characters, so the D/K# signal is driven low when the Start and End characters are inserted (see Table 11-1 on page 386 for a list of Control characters).

# Chapter 11: Physical Layer - Logical (Gen1 and Gen2 Only)

Figure 11-5: Physical Layer Transmit Logic Details (Gen1 and Gen2 Only)

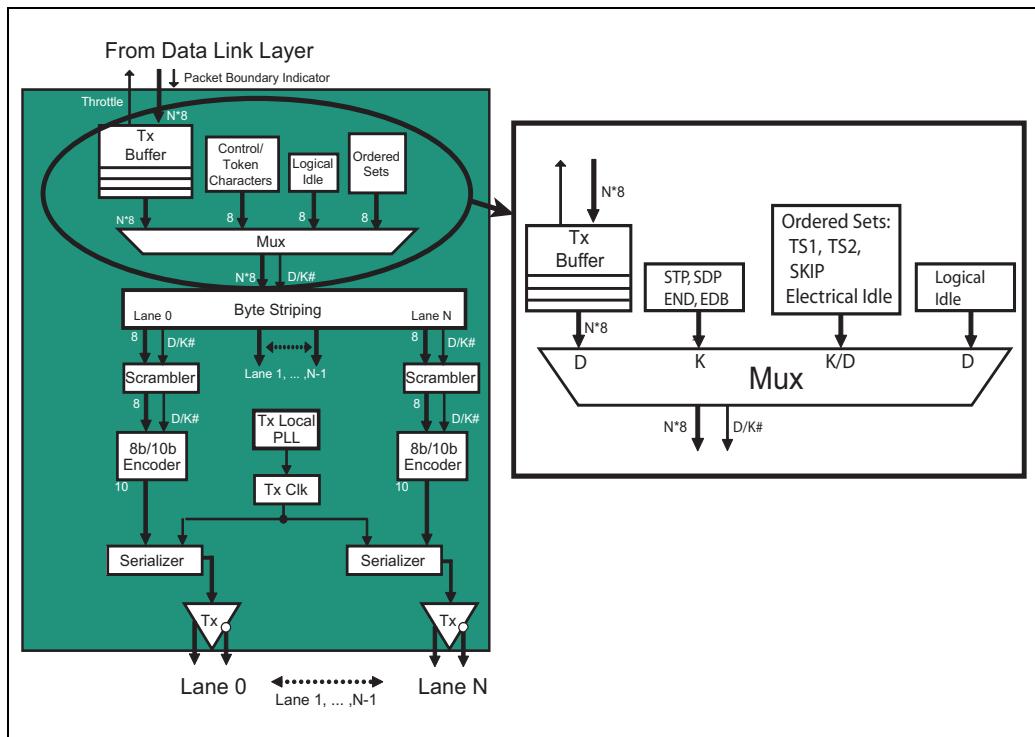


- **Ordered Sets.** As mentioned earlier, control characters are only used by the Physical Layer and are not seen by the higher layers. Some communication across the Link is necessary to initiate and maintain Link operation, and that is accomplished by exchanging Ordered Sets. Every ordered set starts with a K character called a comma (COM), and contains other K or D characters depending on the type of Order Set to be delivered. Ordered Sets are always aligned on four byte boundaries and are transmitted during a variety of circumstances including:
  - Error recovery, initiating events (such as Hot Reset), or exit from low-power states. In these cases, the Training Sequence 1 and 2 (TS1 and TS2) ordered sets are exchanged across the Link.
  - At periodic intervals, the mux inserts the SKIP ordered set pattern to facilitate clock tolerance compensation in the receiver. For a detailed description of this process, refer to “Clock Compensation” on 391.

# PCI Express Technology

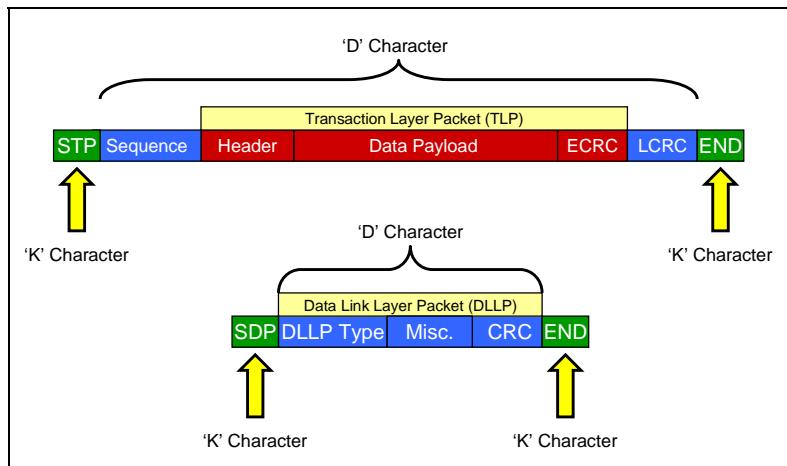
- When a device wants to place its transmitter in the Electrical Idle state, it must inform the remote receiver at the other end of the Link. The mux inserts an **Electrical Idle ordered set** to accomplish this.
- When a device wants to change the Link power state from L0s low power state to the L0 full-on power state, it sends a set of **Fast Training Sequence (FTS)** ordered sets to the receiver. The receiver uses this ordered set to re-synchronize its PLL to the transmitter clock.
- **Logical Idle Sequence.** When there are no packets ready to transmit and no ordered sets to send, the link is logically idle. In order to keep the receiver PLL locked on to the transmitter's frequency, it's important that the transmitter keep sending something, so Logical Idle characters are inserted for that case. Logical Idle is very simple, and consists of nothing more than a string of Data 00h characters.

Figure 11-6: Transmit Logic Multiplexer



## Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

Figure 11-7: TLP and DLLP Packet Framing with Start and End Control Characters



### Byte Striping (for Wide Links)

The next step shown in our example is Byte Stripping, although this is only needed if the port implements more than one Lane (called a wide Link). Stripping means that each consecutive outbound character in a character stream is routed onto consecutive Lanes. The number of Lanes used is configured during the Link training process based on what is supported by both devices that share the Link.

Three examples of byte striping are illustrated in the following diagrams. In Figure 11-8 on page 372, a single-lane link (x1) is shown. This is not a very interesting case, since the packet enters the Physical Layer a byte at a time and goes out the same way, but illustrates the way the sequence of characters will be drawn.

Figure 11-9 on page 372 shows the incoming Dword packets from the multiplexer. Each byte is directed to the corresponding lanes. Finally, Figure 11-10 on page 373 illustrates an eight-lane (x8) link. In this example, two Dwds are required to populate all 8 lanes. This requires the Dwd to arrive at twice the rate as the previous example. The format of the data being sent across each lane is described in the sections that follow.

# PCI Express Technology

---

Figure 11-8: x1 Byte Striping

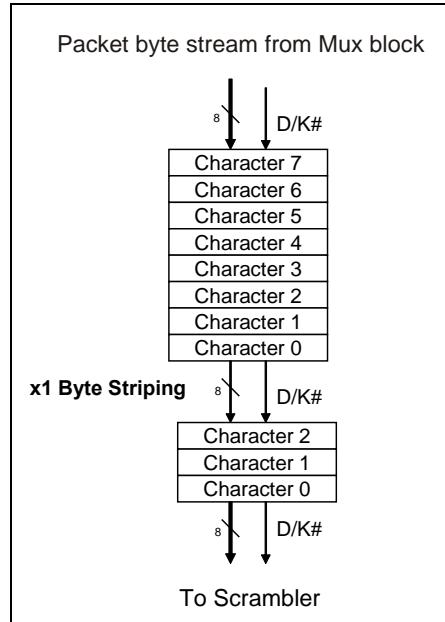
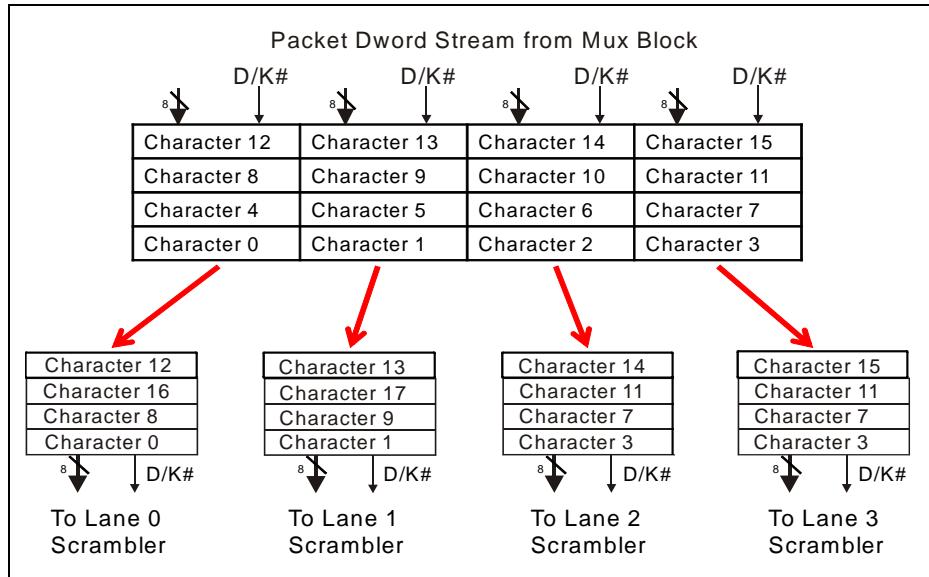
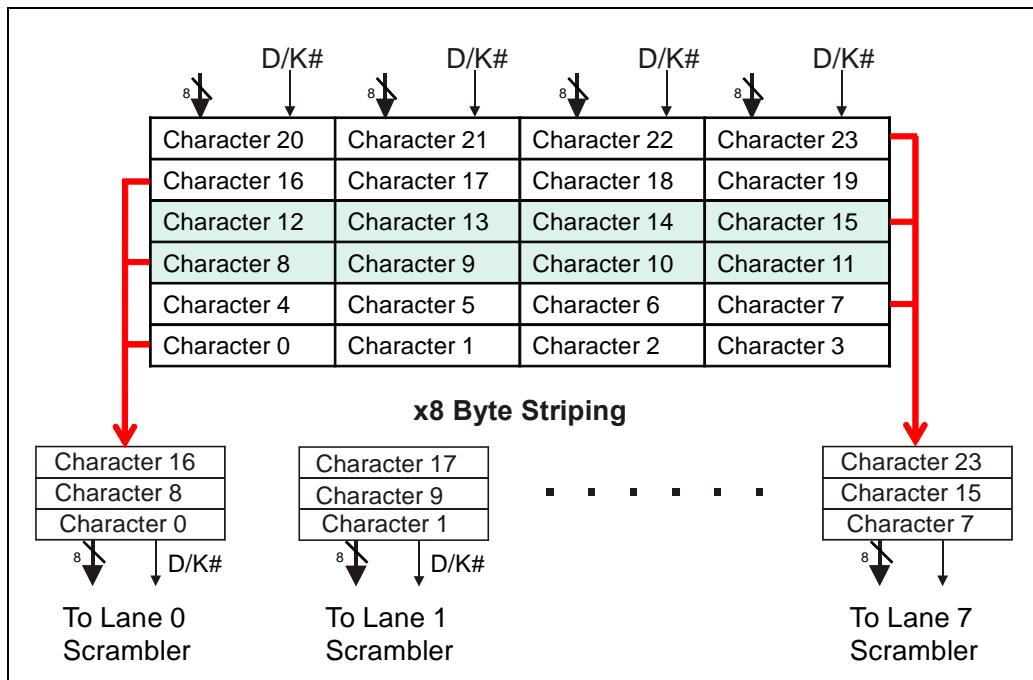


Figure 11-9: x4 Byte Striping



# Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

Figure 11-10: x8 Byte Striping with DWord Parallel Data



## Packet Format Rules

### General Rules

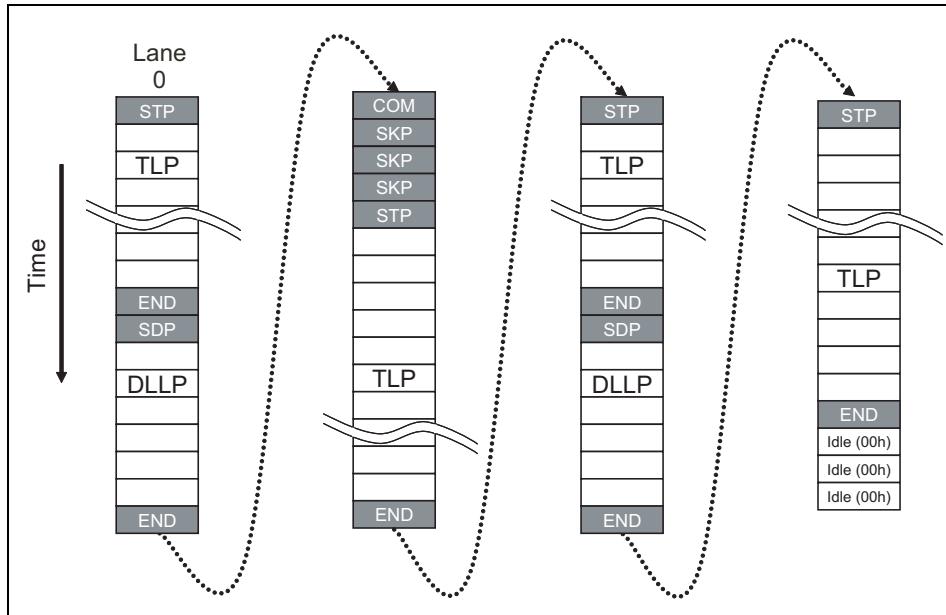
- The total packet length (including Start and End characters) of each packet is always a multiple of four characters. This is a natural extension of the fact that the data length is measured in dwords.
- TLPs start with the STP character and finish with either an END or EDB character.
- DLLPs start with SDP, terminate with the END character, and are exactly 8 characters long (SDP + 6 characters + END)
- STP and SDP characters are placed on Lane 0 when starting the transmission of a packet after the transmission of Logical Idles. In other cases, they may start on a Lane number divisible by 4.
- The receiver's Physical Layer is allowed to watch for violation of these rules and may report them as Receiver Errors to the Data Link Layer.

# PCI Express Technology

## Example: x1 Format

The example shown in Figure 11-11 on page 374 illustrates the format of packets transmitted over a x1 link (a link with only one lane operational). A sequence of packets is shown interspersed with one SKIP Ordered Set. Logical Idles are shown at the end to represent the case when the transmitter has no more packets to send and uses idle characters as filler.

Figure 11-11: x1 Packet Format



## x4 Format Rules

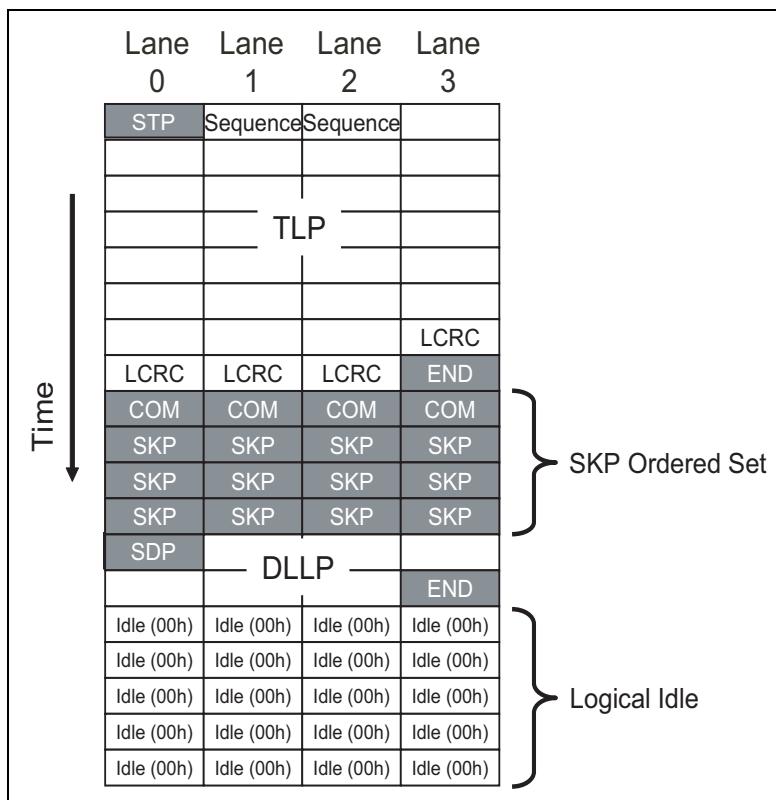
- STP and SDP characters are always sent on Lane 0.
- END and EDB characters are always sent on Lane 3.
- When an ordered set such as the SKIP is sent, it must appear on all lanes simultaneously.
- When Logical Idles are transmitted, they must be sent on all lanes simultaneously.
- Any violation of these rules may be reported as a Receiver Error to the Data Link Layer.

## Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

### Example x4 Format

The example shown in Figure 11-12 on page 375 illustrates the format of packets sent over a x4 Link (link with four data lanes operational). The illustration shows one TLP followed by a SKIP ordered set transmitted on all Lanes for receiver clock compensation. Next is a DLLP, followed by Logical Idle on all lanes. This example highlights that the packets are always multiples of 4 characters because the start character always appears in lane 0 and the end character is always in lane 3. It also illustrates that ordered sets must appear on all the lanes simultaneously.

Figure 11-12: x4 Packet Format



## Large Link-Width Packet Format Rules

The following rules apply when a packet is transmitted over a x8, x12, x16, or x32 Link:

- STP/SDP characters are always sent on Lane 0 when transmission starts after a period during which Logical Idles are transmitted. After that, they may only be sent on Lane numbers divisible by 4 when sending back-to-back packets (Lane 4, 8, 12, etc.).
- END/EDB characters are sent on Lane numbers divisible by 4 and then minus one (Lane 3, 7, 11, etc.).
- If a packet doesn't end on the last Lane of the Link and there are no more packets ready to go, PAD Symbols are used as filler on the remaining lane numbers. Logical Idle can't be used for this purpose because it must appear on all Lanes at the same time.
- Ordered sets must be sent on all lanes simultaneously.
- Similarly, logical idles must be sent on all lanes when they are used.
- Any violation of these rules may be reported as a Receiver Error to the Data Link Layer.

## x8 Packet Format Example

The example shown in Figure 11-13 on page 377 illustrates the format of packets transmitted over a x8 link. The illustration shows a TLP followed by a SKIP ordered set, a DLLP, and finally a TLP that ends on Lane 3. At that point, the transmitter has no more packets ready to send but the current packet doesn't extend to include all the available lanes. One might expect the extra lanes to be filled with Logical Idle, but it won't work here because idles must appear on all lanes at the same time. So another fill character is needed, and the spec writers chose to use the PAD control character here. The only other place that PAD is used is during the training process. Finally, since there are still no more packets to send, Logical Idles are sent on all the lanes.

## Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

Figure 11-13: x8 Packet Format

Lane 0	Lane 1	Lane 2	Lane 3	Lane 4	Lane 5	Lane 6	Lane 7
Idle (00h)							
STP	Sequence	Sequence					
				TLP			
				LCRC	LCRC	LCRC	LCRC
COM							
SKP							
SKP							
SKP							
SDP			DLLP				END
STP	Sequence	Sequence					
				TLP			LCRC
LCRC	LCRC	LCRC	END	PAD	PAD	PAD	PAD
Idle (00h)							
Idle (00h)							

---

### Scrambler

The next step in our example is scrambling, as shown in Figure 11-5 on page 369, which is intended to prevent repetitive patterns in the data stream. Repetitive patterns create “pure tones” on the link, meaning a consistent frequency caused by the pattern that generates more than the usual noise, or EMI. Reducing this problem by spreading this energy over a wider frequency range is the primary goal of scrambling. In addition, though, scrambled transmission on one Lane also reduces interference with adjacent Lanes on a wide Link. This “spatial frequency de-correlation”, or reduction of crosstalk noise, helps the receiver on each lane to distinguish the desired signal.

# PCI Express Technology

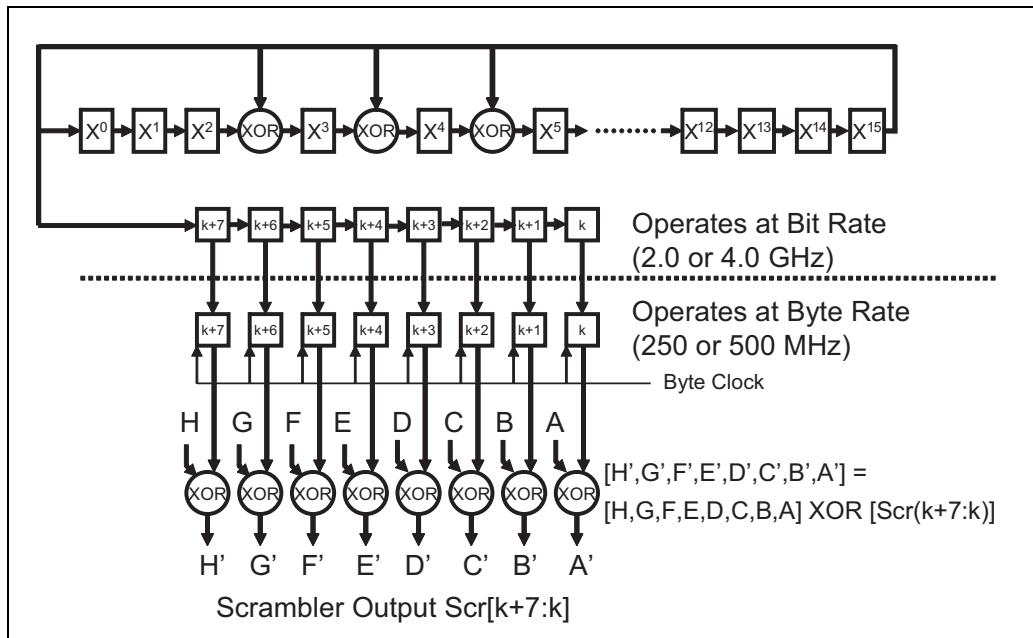
To help the receiver maintain synchronization with the scrambled sequence, control characters do not get scrambled and are thus recognizable even if the scramblers get out of sync. In addition, the arrival of the COM control character (K28.5) reinitializes the scramblers on both ends of the Link each time it arrives and thus re-synchronizes them.

## Scrambler Algorithm

The scrambler described in the spec is shown in Figure 11-14 on page 378. It's made of a 16-bit Linear Feedback Shift Register (LFSR) with feedback points that implement the following polynomial:

$$G(x) = X^{16} + X^5 + X^4 + X^3 + 1$$

Figure 11-14: Scrambler



The LFSR is clocked at 8 times the frequency of the clock feeding the data bytes, and its output is clocked into an 8-bit register that is XORed with the 8-bit data characters to form the scrambled data output.

# **Chapter 11: Physical Layer - Logical (Gen1 and Gen2)**

---

## **Some Scrambler implementation rules:**

- On a multi-Lane Link implementation, Scramblers associated with each Lane must operate in concert, maintaining the same simultaneous value in each LFSR.
- Scrambling is applied to 'D' characters only, meaning those associated with TLP and DLLPs and the Logical Idle (00h) characters. However, those 'D' characters that are within the TS1 and TS2 ordered sets are not scrambled.
- Scrambling is never applied to 'K' characters and characters within ordered sets, such as TS1, TS2, SKIP, FTS and Electrical Idle. These characters bypass the scrambler logic. One reason for this is to ensure they'll still be recognizable by the receiver even if the scramblers somehow get out of sequence.
- Compliance Pattern characters (used for testing) are also not scrambled.
- The COM character, a control character that does not get scrambled, is used to reinitialize the LFSR to FFFFh at both the transmitter and receiver.
- Except for the COM character, the LFSR normally will serially advance eight times for every D or K character sent, but it does not advance on SKP characters associated with the SKIP ordered set. The reason is that a receiver may add or delete SKP Symbols to perform clock tolerance compensation. Changing the number of characters in the receiver compared to the number that were sent would cause the value in the receiver LFSR to lose synchronization with the transmitter LFSR value if they were not ignored.

## **Disabling Scrambling**

Scrambling is enabled by default, but the spec allows it to be disabled for test and debug purposes. That's because testing may require control of the exact bit pattern sent and, since the hardware handles scrambling, there's no reasonable way for the software to be able to force a specific pattern. No specific software mechanism is defined by which to instruct the Physical Layer to disable scrambling, so this has to be a design-specific implementation.

If scrambling is disabled by a device, this gets communicated to the neighboring device by sending at least two TS1s and TS2s that have the appropriate bit set in the control field as described in "Configuration State" on page 539. In response, the neighboring device also disables its scrambling.

## 8b/10b Encoding

### General

The first two generations of PCIe use 8b/10b encoding. Each Lane implements an 8b/10b Encoder that translates the 8-bit characters into 10-bit Symbols. This coding scheme was patented by IBM in 1984 and is widely used in many serial transports today, such as Gigabit Ethernet and Fibre Channel.

### Motivation

Encoding accomplishes several desirable goals for serial transmission. Three of the most important are listed here:

- **Embedding a Clock into the Data.** Encoding ensures that the data stream has enough edges in it to recover a clock at the Receiver, with the result that a distributed clock is not needed. This avoids some limitations of a parallel bus design, such as flight time and clock skew. It also eliminates the need to distribute a high-frequency clock that would cause other problems like increased EMI and difficult routing.

As an example of this process, Figure 11-15 on page 381 shows the encoding results of the data byte 00h. As can be seen, this 8-bit character that had no transitions converts to a 10-bit Symbol with 5 transitions. The 8b/10b guarantees enough edges to ensure the “run length” (sequence of consecutive ones or zeros) in the bit stream to no more than 5 consecutive bits under any conditions.

- **Maintaining DC Balance.** PCIe uses an AC-coupled link, placing a capacitor serially in the path to isolate the DC part of the signal from the other end of the Link. This allows the Transmitter and Receiver to use different common-mode voltages and makes the electrical design easier for cases where the path between them is long enough that they’re less likely to have exactly the same reference voltages. That DC value, or common-mode voltage, can change during run time because the line charges up when the signal is driven. Normally, the signal changes so quickly that there isn’t time for this to cause a problem but, if the signal average is predominantly one level or the other, the common-mode value will appear to drift. Referred to as “DC Wander”, this drifting voltage degrades signal integrity at the Receiver. To compensate, the 8b/10b encoder tracks the “disparity” of the last Symbol that was sent. Disparity, or inequality, simply indicates whether the previous Symbol had more ones than zeros (called positive disparity), more zeros than ones (negative disparity), or a balance of ones and zeros (neutral

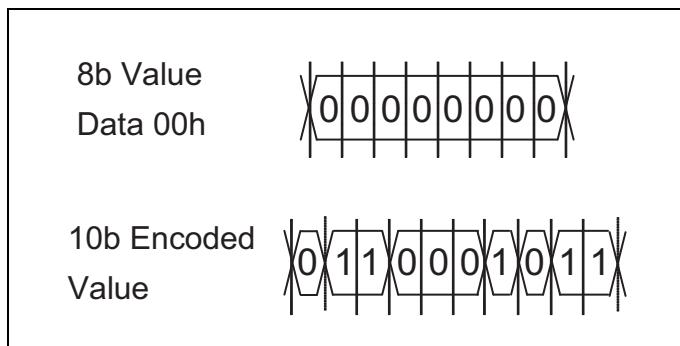
## Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

disparity). If the previous Symbol had negative disparity, for example, the next one should balance that by using more ones.

- **Enhancing Error Detection.** The encoding scheme also facilitates the detection of transmission errors. For a 10-bit value, 1024 codes are possible, but the character to be encoded only has 256 unique codes. To maintain DC balance the design uses two codes for each character, and chooses which one based on the disparity of the last Symbol that was sent, so 512 codes would be needed. However, many of the neutral disparity encodings have the same values (D28.5 is one example), so not all 512 are used. As a result, more than half the possible encodings are not used and will be considered illegal if seen at a Receiver. If a transmission error does change the bit pattern of a Symbol, there's a good chance the result would be one of these illegal patterns that can be recognized right away. For more on this see the section titled, "Disparity" on page 383.

The major disadvantage of 8b/10b encoding is the overhead it requires. The actual transmission performance is degraded by 20% from the Receiver's point of view because 10 bits are sent for each byte, but only 8 useful bits are recovered at the receiver. This is a non-trivial price to pay but is still considered acceptable to gain the advantages mentioned.

Figure 11-15: Example of 8-bit Character 00h Encoding



### Properties of 10-bit Symbols

As described in the literature on 8b/10b coding, the design isn't strictly 8 bits to 10 bits. Instead, it's really a 5-to-6 bit encoding followed by a 3-to-4 bit encoding. The sub-blocks are internal to the design but their existence helps to explain some of the properties for a legal Symbol, as listed below. A Symbol that doesn't follow these properties is considered invalid.

# PCI Express Technology

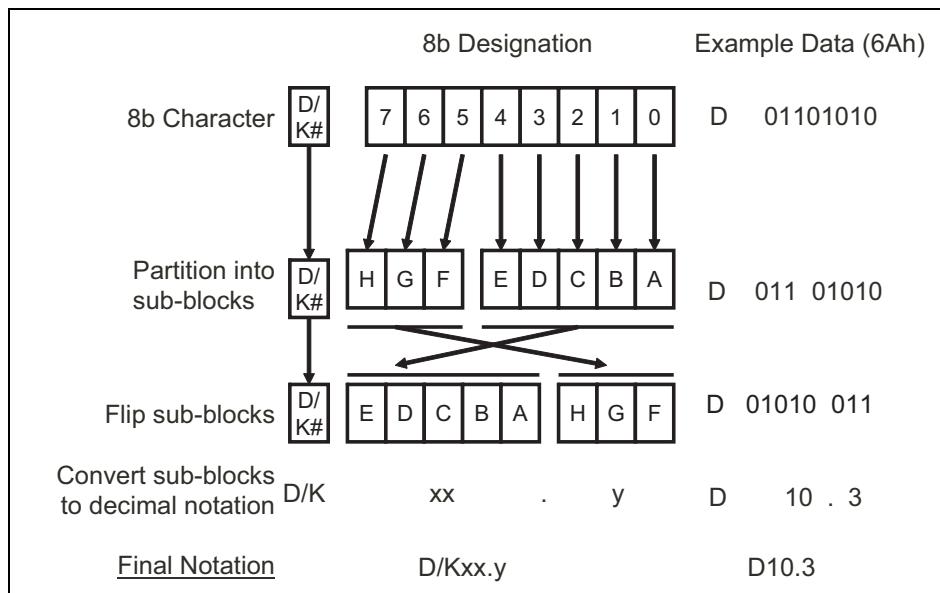
- The bit stream never contains more than five continuous 1s or 0s, even from the end of one Symbol to beginning of the next.
- Each 10-bit Symbol contains:
  - Four 0s and six 1s (not necessarily contiguous), or
  - Six 0s and four 1s (not necessarily contiguous), or
  - Five 0s and five 1s (not necessarily contiguous).
- Each 10-bit Symbol is subdivided into two sub-blocks: the first is six bits wide and the second is four bits wide.
  - The 6-bit sub-block contains no more than four 1s or four 0s.
  - The 4-bit sub-block contains no more than three 1s or three 0s.

## Character Notation

The 8b/10b uses a special notation shorthand, and Figure 11-16 on page 382 illustrates the steps to arrive at the shorthand for a given character:

1. Partition the character into its 3-bit and 5-bit sub-blocks.
2. Transpose the position of the sub-blocks.
3. Create the decimal equivalent for each sub-block.
4. The character takes the form Dxx.y for Data characters, or Kxx.y for Control characters. In this notation, xx is the decimal equivalent of the 5-bit field, and y is the decimal equivalent of the 3-bit field.

Figure 11-16: 8b/10b Nomenclature



# Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

## **Disparity**

**Definition.** Disparity refers to the inequality between the number of ones and zeros within a 10-bit Symbol and is used to help maintain DC balance on the link. A Symbol with more zeros is said to have a negative (-) disparity, while a Symbol with more ones has a positive (+) disparity. When a Symbol has an equal number of ones and zeros, it's said to have a neutral disparity. Interestingly, most characters encode into Symbols with + or - disparity, but some only encode into Symbols with neutral disparity.

**CRD (Current Running Disparity).** The CRD is the information as to the current state of disparity on the link. Since it's just a single bit it can only be positive or negative and doesn't always change when the next Symbol is sent out. To see how it works, remember that the next Symbol decoded can have negative, neutral, or positive disparity, then consider the following example. If the CRD was positive, an outgoing Symbol with a negative disparity would change it to negative, a neutral disparity would leave it as positive, and a positive disparity would be an error because the CRD is only one bit and can't be made more positive.

The initial state of the CRD (before any characters are transmitted) may not match between the sender and receiver but it turns out that it doesn't matter. When the receiver sees the first Symbol after training is complete, it will check for a disparity error and, if one is found, just change the CRD. This won't be considered an error but simply an adjustment of the CRD to match the receiver and sender. After that, there are only two legal CRD cases: it can remain the same if the new Symbol has neutral disparity, or it can flip to the opposite polarity if the new Symbol has the opposite disparity. What is not legal is for the disparity of the new Symbol to be the same as the CRD. Such an event would be a disparity error and should never occur after the initial adjustment unless an error has occurred.

## **Encoding Procedure**

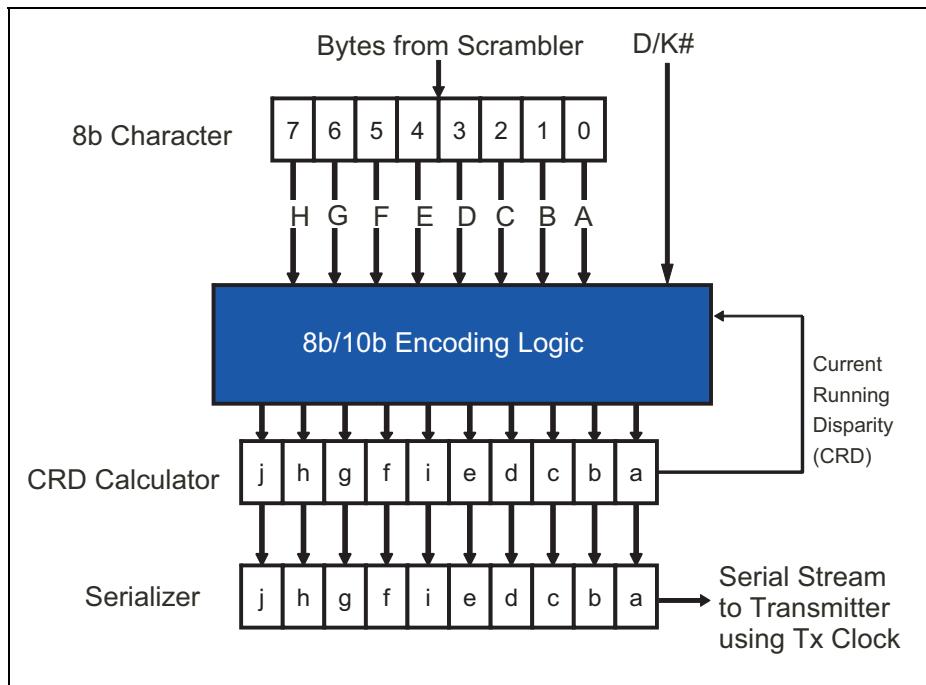
There are different ways that 8b/10b encoding could be accomplished. The simplest approach is probably to implement a look-up table that contains all the possible output values. However, this table can require a comparatively large number of gates. Another approach is to implement the decoder as a logic block, and this is usually the preferred choice because it typically results in a smaller and cheaper solution. The specifics of the encoding logic are described in detail in the referenced literature, so we'll focus here on the bigger picture of how it works instead.

# PCI Express Technology

---

An example 8b/10b block diagram is shown in Figure 11-17 on page 384. A new outgoing Symbol is created based on three things: the incoming character, the D/K# indication for that character, and the CRD. A new CRD value is computed based on the outgoing Symbol and is fed back for use in encoding the next character. After encoding, the resulting Symbol is fed to a serializer that clocks out the individual bits. Figure 11-18 on page 385 shows some sample 8b/10b encodings that will be useful for the example that follows.

Figure 11-17: 8-bit to 10-bit (8b/10b) Encoder



## Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

Figure 11-18: Example 8b/10b Encodings

D or K Character	Hex Byte	Binary Bits HGF EDCBA	Byte Name	Encodes to this if CRD is positive	Encodes to this if CRD is negative
				CRD – abcdei fghj	CRD + abcdei fghj
Data (D)	6A	011 01010	D10.3	010101 1100	010101 0011
Data (D)	1B	000 11011	D27.0	110110 0100	001001 1011
Data (D)	F7	111 10111	D23.7	111010 0001	000101 1110
Control (K)	F7	111 10111	K23.7	111010 1000	000101 0111
Control (K)	BC	101 11100	K28.5	001111 1010	110000 0101

### Example Transmission

Figure 11-19 illustrates the encode and transmission of three characters: the first and second are the control character K28.5 and the third character is the data character D10.3.

In this example the initial CRD is negative so K28.5 encodes into 001111 1010b. This Symbol has positive disparity (more ones than zeros), and causes the CRD polarity to flip to positive. The next K28.5 is encoded into 110000 0101b and has a negative disparity. That causes the CRD this time to flip to negative. Finally, D10.3 is encoded into 010101 1100b. Since its disparity is neutral, the CRD doesn't change in this case but remains negative for whatever the next character will be.

# PCI Express Technology

Figure 11-19: Example 8b/10b Transmission

Use these two characters in the example below:

D/K#	Hex Byte	Binary Bits HGF EDCBA	Byte Name	CRD – abcdei fghj	CRD + abcdei fghj
Control (K)	BC	101 11100	K28.5	001111 1010	110000 0101
Data (D)	6A	011 01010	D10.3	010101 1100	010101 0011

## Example Transmission

	CRD	Character	CRD	Character	CRD	Character	CRD
Character to be transmitted		K28.5 (BCh)		K28.5 (BCh)		D10.3 (6Ah)	
Bit stream transmitted	-	Yields 001111 1010 CRD is +	+	Yields 110000 0101 CRD is -	-	Yields 010101 1100 CRD is neutral	-

Initialized value of CRD is don't care. Receiver can determine from incoming bit stream

## Control Characters

The 8b/10b encoding provides several special characters for Link management and Table 11-1 on page 386 shows their encoding.

Table 11-1: Control Character Encoding and Definition

Character Name	8b/10b Name	Description
COM	K28.5	First character in any ordered set. Also used by Rx to achieve Symbol lock during training.
PAD	K23.7	Packet filler
SKP	K28.0	Used in SKIP ordered set for Clock Tolerance Compensation

## Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

---

Table 11-1: Control Character Encoding and Definition (Continued)

Character Name	8b/10b Name	Description
STP	K27.7	Start of a TLP
SDP	K28.2	Start of a DLLP
END	K29.7	End of Good Packet
EDB	K30.7	End of a bad or ‘nullified’ TLP.
FTS	K28.1	Used to exit from L0s low power state to L0
IDL	K28.3	Used to place Link into Electrical Idle state
EIE	K28.7	Part of the Electrical Idle Exit Ordered Set sent prior to bringing the Link back to full power for speeds higher than 2.5 GT/s

- **COM** (Comma): One of the main functions of this is to be the first Symbol in the physical layer communications called ordered sets (see “Ordered sets” on page 388). It has an interesting property that makes both of its Symbol encodings easily recognizable at the receiver: they start with two bits of one polarity followed by five bits of the opposite polarity (001111 1010 or 110000 0101). This property is especially helpful for initial training, when the receiver is trying to make sense of the string of bits coming in, because it helps the receiver lock onto the incoming Symbol stream. See “Link Training and Initialization” on page 405 for more on how this works.
- **PAD**: On a multi-Lane Link, if a packet to be sent doesn’t cover all the available lanes and there are no more packets ready to send, the PAD character is used to fill in the remaining Lanes.
- **SKP** (Skip): This is used as part of the SKIP ordered set that is sent periodically to facilitate clock tolerance compensation.
- **STP** (Start TLP): Inserted to identify the start of a TLP.
- **SDP** (Start DLLP): Inserted to identify the start of a DLLP.
- **END**: Appended to identify the end of an error-free TLP or DLLP.
- **EDB** (EnD Bad): Inserted to identify the end of a TLP that a forwarding device (such as a switch) wishes to ‘nullify’. This case can arise when a switch using the “cut-through mode” forwards a packet from an ingress port to an egress port without buffering the whole packet first. Any error detected during the forwarding process creates a problem because a portion of the packet is already being delivered before the packet can be checked for

errors. To handle this case, the switch must cancel the one that's already in route to the destination. This is accomplished by nullifying it: ending the packet with EDB and inverting the LCRC from what it should have been. When a receiver sees a nullified packet, it discards the packet and does not return an ACK or NAK. (See the "Example of Cut-Through Operation" on page 356.)

- **FTS** (Fast Training Sequence): Part of the FTS ordered set sent by a device to recover a link from the L0s standby state back to the full-on L0 state.
- **IDL** (Idle): Part of the Electrical Idle ordered set sent to inform the receiver that the Link is transitioning into a low power state.
- **EIE** (Electrical Idle Exit): Added in the PCIe 2.0 spec and used to help an electrically-idle link begin the wake up process.

## Ordered sets

**General.** Ordered Sets are used for communication between the Physical Layers of Link partners and may be thought of as Lane management packets. By definition they are a series of characters that are not TLPs or DLLPs. For Gen1 and Gen2 they always begin with the COM character. Ordered Sets are replicated on all Lanes at the same time, because each Lane is technically an independent serial path. This also allows Receivers to verify alignment and de-skewing. Ordered Sets are used for things like Link training, clock tolerance compensation, and changing Link power states.

**TS1 and TS2 Ordered Set (TS1OS/TS2OS).** Training sequences one and two are used for Link initialization and training. They allow the Link partners to achieve bit lock and Symbol lock, negotiate the link speed, and report other variables associated with Link operation. They are described in more detail in the section titled "TS1 and TS2 Ordered Sets" on page 510.

**Electrical Idle Ordered Set (EIOS).** A Transmitter that wishes to go to a lower-power link state sends this before ceasing transmission. Upon receipt, Receivers know to prepare for the low power state. The EIOS consists of four Symbols: the COM Symbol followed by three IDL Symbols. Receivers detect this Ordered Set and prepare for the Link to go into Electrical Idle by ignoring input errors until exiting from Electrical Idle. Shortly after sending EIOS, the Transmitter reduces its differential voltage to less than 20mV peak.

**FTS Ordered Set (FTSOS).** A Transmitter sends the proper number of these (the minimum number was given by the Link neighbor during training) to take a Link from the low-power L0s state back to the fully-operational L0 state. The receiver detects the FTSs, recognizes that the Link is

## **Chapter 11: Physical Layer - Logical (Gen1 and Gen2)**

---

exiting from Electrical Idle, and uses them to recover Bit and Symbol Lock. The FTS Ordered Set consists of four Symbols: the COM Symbol followed by three FTS Symbols.

**SKP Ordered Set (SOS).** This consists of four Symbols: the COM Symbol followed by three SKP Symbols. It's transmitted at regular intervals and is used for Clock Tolerance Compensation as described in "Clock Compensation" on page 391 and "Receiver Clock Compensation Logic" on page 396. Basically, the Receiver evaluates the SOS and internally adds or removes SKP Symbols as needed to prevent its elastic buffer from under-flowing or over-flowing.

**Electrical Idle Exit Ordered Set (EIEOS).** Added in the PCIe 2.0 spec, this Ordered Set was defined to provide a lower-frequency sequence required to exit the electrical idle Link state. The EIEOS for 8b/10b encoding, uses repeated K28.7 control characters to appear as a repeating string of 5 ones followed by 5 zeros. This low frequency string produces a low-frequency signal that allows for higher signal voltages that are more readily detected at the receiver. In fact, the spec states that this pattern guarantees that the Receiver will properly detect an exit from Electrical Idle, something that scrambled data cannot do. For details on electrical idle exit, refer to the section "Electrical Idle" on page 736.

---

### **Serializer**

The 8b/10b encoder on each lane feeds a serializer that clocks the Symbols out in bit order (see Figure 11-17 on page 384), with the least significant bit (a) shifted out first and the most significant bit (j) shifted out last. For each lane, the Symbols will be supplied to the serializer at either 250MHz or 500MHz to support a serial bit rate 10 times faster than that at 2.5 GHz or 5.0 GHz.

---

### **Differential Driver**

The differential driver that actually sends the bit stream onto the wire uses NRZ encoding. NRZ simply means that there are no special or intermediate voltage levels used. Differential signalling improves signal integrity and allows for both higher frequencies and lower voltages. Details regarding the electrical characteristics of the driver are discussed in the section "Transmitter Voltages" on page 462.

## Transmit Clock (Tx Clock)

The serialized output on each Lane is clocked out by the Tx Clock signal. As mentioned earlier, the clock frequency must be accurate to +/-300ppm around the center frequency (600ppm total variation). There are two options regarding the source of this clock. It can be generated internally or derived from an external reference that may optionally be available. The PCIe spec for peripheral cards includes the definition of a 100MHz reference clock supplied by the system board for this purpose. This reference clock is multiplied internally to derive the local clock that drives the internal logic and the Tx clock used by the serializer.

---

## Miscellaneous Transmit Topics

### Logical Idle

In order to keep the receiver's PLL from drifting, something must be transmitted during periods when there are no TLPs, DLLPs or ordered sets to transmit, and it is logical idle characters that are injected into the character flow during these times. Some properties of the logical idle character:

- It's an 8-bit Data character with a value of 00h.
- When sent, it goes on all Lanes at the same time and the Link is said to be in the logical idle state (not to be confused with electrical Idle—the state when the output driver stops transmitting altogether and the receiver PLL loses synchronization).
- The logical idle character is scrambled, but a receiver can distinguish it from other data because it occurs outside of a packet framing context (i.e.: after an END or EDB, but before an STP or SDP).
- It is 8b/10b encoded.
- During logical idle transmission, SKIP ordered sets are still sent periodically.

### Tx Signal Skew

Understandably, the transmitter should introduce a minimal skew between lanes to leave as much Rx skew budget as possible for routing and other variations. The spec lists the Tx skew values as 500ps + 2 UI for Gen1, 500ps + 4UI for Gen2, and 500ps + 6 UI for Gen3. Recalling that UI (unit interval) represents one bit time on the Link, this works out as shown in Table 11-2 below.

# Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

*Table 11-2: Allowable Transmitter Signal Skew*

Spec Version	Allowable Tx Skew
Gen1	1300 ps
Gen2	1300 ps
Gen3	1250 ps

## Clock Compensation

**Background.** High-speed serial transports like PCIe have a particular clock problem to solve. The receiver recovers a clock from the incoming bit stream and uses that to latch in the data bits, but this recovered clock is not synchronized with the receiver's internal clock and at some point it has to begin clocking the data with its own internal clock. Even if they have an optional common external reference clock, the best they can do is to generate an internal clock within a specified tolerance of the desired frequency. Consequently, one of the clocks will almost always have a slightly higher frequency than the other. If the transmitter clock is faster, the packets will be arriving faster than they can be taken in. To compensate, the transmitter must inject some "throw-away characters" in the bit stream that the receiver can discard if it proves necessary to avoid a buffer over-run condition. For PCIe, these characters which can be deleted take the form of the SKIP ordered set, which consists of a COM character followed by three SKP characters (see Figure 11-20). For more detail on this topic, refer to "Receiver Clock Compensation Logic" on page 396).

**SKIP ordered set Insertion Rules.** A transmitter is required to send SKIP ordered sets on a periodic basis, and the following rules apply:

- The SKIP ordered set must be scheduled for insertion between 1180 and 1538 Symbol times (a Symbol time is the time required to send one Symbol and is 10 bit times, so at 2.5 GT/s, a Symbol time is 4ns and at 5.0 GT/s, it's 2ns).
- They are only inserted on packet boundaries (nothing is allowed to interrupt a packet) and must go simultaneously on all Lanes. If a packet is already in progress the SKP Ordered Set will have to wait. The maximum possible packet size would require more than 4096 Symbol times, though, and during that time several SKIP ordered sets should have

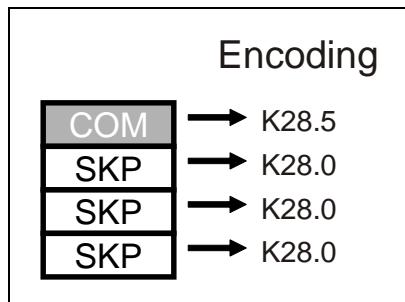
# PCI Express Technology

---

been sent. This case is handled by accumulating the SKIPs that should have gone out and injecting them all at the next packet boundary.

- Since this ordered set must be transmitted on all Lanes simultaneously, a multi-lane link may need to add PAD characters on some Lanes to allow the ordered set to go on all Lanes simultaneously (see Figure 11-13 on page 377).
- During low-power link states, any counters used to schedule SKIP ordered sets must be reset. There's no need for them when the transmitter isn't signaling, and it wouldn't make sense to wake up the link to send them.
- SKIP ordered sets must not be transmitted while the Compliance Pattern is in progress.

Figure 11-20: SKIP Ordered Set



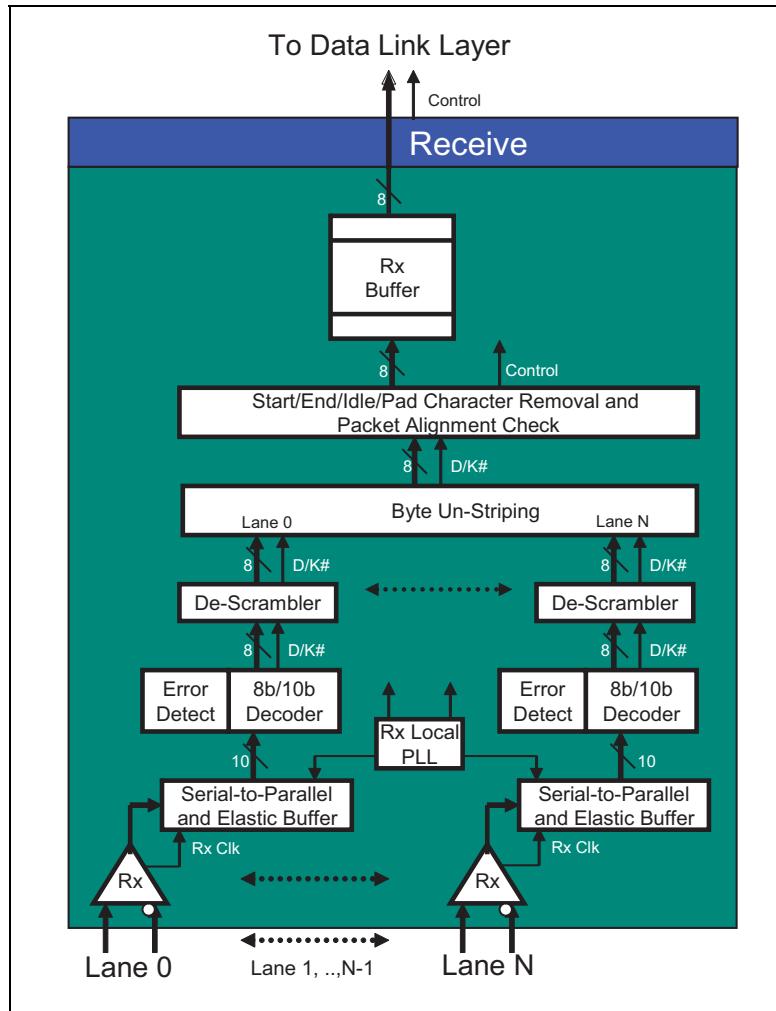
---

## Receive Logic Details (Gen1 and Gen2 Only)

Figure 11-21 shows the receiver logic of the Logical Physical Layer. This section describes packet processing from the time the data is received serially on each lane until the packet byte stream is clocked into the Data Link Layer.

# Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

Figure 11-21: Physical Layer Receive Logic Details (Gen1 and Gen2 Only)



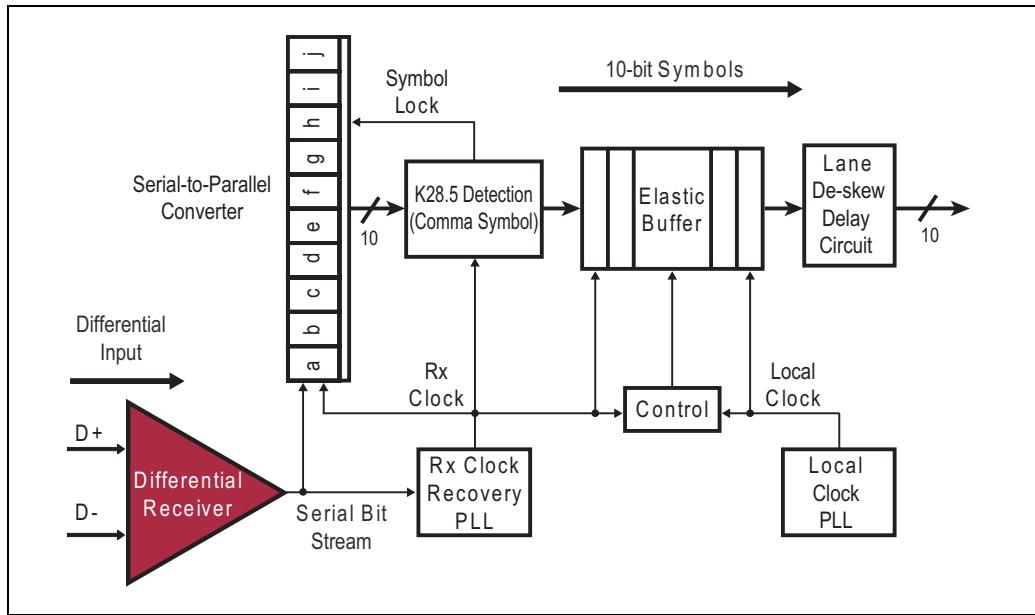
## Differential Receiver

The first parts of the receiver logic are shown in Figure 11-22, including the differential input buffer for each lane. The buffer senses peak-to-peak voltage differences and determines whether the difference represents a logical one or zero.

# PCI Express Technology

For a detailed discussion of receiver characteristics, see section “Receiver Characteristics” on page 492.

Figure 11-22: Receiver Logic’s Front End Per Lane



## Rx Clock Recovery

### General

Next the receiver generates an Rx Clock from the data bit transitions in the input data stream, probably using a PLL. This recovered clock has the same frequency (2.5 or 5.0 GHz) as that of the Tx Clock that was used to clock the bit stream onto the wire. The Rx Clock is used to clock the inbound bit stream into the deserializer. The deserializer has to be aligned to the 10-bit Symbol boundary (a process called achieving Symbol lock), and then its Symbol stream output is clocked into the elastic buffer with a version of the Rx Clock that's been divided by 10. Even though both must be accurate to within +/-300ppm of the center frequency, the Rx Clock is probably a little different from the Local Clock and if so, compensation is needed.

# **Chapter 11: Physical Layer - Logical (Gen1 and Gen2)**

---

## **Achieving Bit Lock**

Recall that the 8b/10b encoding scheme guarantees the inbound serial Symbol stream will contain frequent transitions. The receiver PLL uses those transitions to create an Rx Clock that is synchronized with the Tx Clock that was used to clock the bit stream out of the transmitter. When the receiver locks on to the Tx Clock frequency, the receiver is said to have achieved “**Bit Lock**”.

During Link training, the transmitter sends a long series of TS1 and TS2 ordered sets to the receiver, which then uses the bit transitions in them to achieve Bit Lock. There are enough transitions on the Link during normal operation for the receiver to maintain Bit Lock after that.

## **Losing Bit Lock**

If the Link is put in a low power state (such as L0s or L1) in which packet transmission ceases, the receiver will lose synchronization. To avoid having the error circuit see this as an error, the transmitter sends an electrical Idle ordered set (EIOS) before going to the lower power state to tell the receiver to de-gate its input.

## **Regaining Bit Lock**

When the transmitter is ready to wake the Link from the L0s state, it sends a specific number FTS ordered sets (the actual number is design specific) and the receiver uses these to regain bit and Symbol lock. A relatively small number of FTSs are needed to recover and so the recovery latency is short. Because the Link is in the L0s state for a short time, the receiver PLL does not usually drift too far from the Tx Clock before it begins to receive the FTSs. If the Link was instead in the L1 low power state and the transmitter instead starts transmitting TS1OSs. This results in the Link getting re-trained and wakeup time is longer than L0s wakeup time. Should the Link have a more serious error and the Ack/Nak mechanism be unsuccessful in error recovery after four attempts of retrying the TLPs, the Data Link Layer signals the Physical Layer to re-training the Link. Here again, Bit Lock is re-established during the re-training process.

---

## **Deserializer**

### **General**

The incoming data is clocked into each Lane’s deserializer (serial-to-parallel converter) by the Rx clock (see Figure 11-22 on page 394). The 10-bit Symbols produced are clocked into the Elastic Buffer using a divided-by-10 version of the Rx Clock.

## Achieving Symbol Lock

When the receive logic starts receiving a bit stream, it is JABOB (just a bunch of bits) with no markers to differentiate Symbols or any boundaries. The receive logic must have a way to find the start and end of a 10-bit Symbol, and the Comma (COM) Symbol serves this purpose.

The 10-bit encoding of the COM Symbol contains two bits of one polarity followed by five bits of the opposite polarity (001111010b or 1100000101b), making it easily detectable. Recall that the COM Control character, like all other Control characters, is also not scrambled by the transmitter, and that ensures that the desired sequence will be seen at the receiver. Upon detection of the COM, the logic knows that the next bit received will be the first bit of the next 10-bit Symbol. At that point, the deserializer is said to have achieved '**Symbol Lock**'.

The COM Symbol is used to achieve Symbol Lock as follows:

- During Link training when the Link is first established or when re-training is needed, and TS1 and TS2 ordered sets are transmitted.
- When FTS ordered sets are sent to inform the receiver to change the state of the Link from L0s to L0.

---

## Receiver Clock Compensation Logic

### Background

We've observed before that the clocks used by the transmitter and receiver on either end of a link aren't required to have exactly the same frequencies. This will be the case whenever the link doesn't use a common reference clock and introduces the problem that one of them is running slightly faster than the other. The only requirement is that both clocks must be within  $\pm 300$  ppm (parts per million) of the center frequency. Since one could be +300 ppm and the other could be -300 ppm in the worst case, the worst separation between them could be 600ppm. That difference translates into a gain or loss of one Symbol clock every 1666 clocks. Once the Link is trained, the receive clock (Rx Clock) in the receiver is the same as the transmit clock (Tx Clock) at the other end of the Link (because the receive clock is derived from the bit stream).

# **Chapter 11: Physical Layer - Logical (Gen1 and Gen2)**

---

## **Elastic Buffer's Role**

To compensate for that worst-case frequency difference, an elastic buffer (see Figure 11-22 on page 394) is built into the receive path. Received Symbols are clocked into it using the recovered clock and clocked out using the receiver's local clock. The Elastic Buffer compensates for the frequency difference by adding or removing SKP Symbols. When a SKIP ordered set arrives, logic watching the status of the elastic buffer makes an evaluation. If the local clock is running faster, Symbols are being clocked out faster than they're coming in, so the buffer will be approaching an underflow condition. The logic will compensate for this by appending an extra SKIP Symbol to the ordered set when it arrives to quickly refill the buffer. On the other hand, if the recovered clock is running faster, the buffer will be approaching an overflow condition and the logic will compensate for that by deleting one of the SKIP Symbols to quickly drain the buffer. These actions will make up for difference in rates of arrival and consumption of the Symbols and prevent any confusion or loss of data.

The transmitter periodically sends the SKIP ordered sets for this purpose. As the name implies, the SKIP characters are really disposable characters. Deleting or adding a SKIP Symbol prevents a buffer overflow or underflow in the elastic buffer and then they get discarded along with all the other control characters when the Symbols are forwarded to the next layer. Consequently, they use a little bandwidth but don't otherwise affect the flow of packets at all.

Although lost Symbols due to an Elastic Buffer overflow or underflow is an error condition, it's optional for receivers to check for this. If they do, and this situation occurs, a Receiver Error will be indicated to the Data Link Layer.

The transmitter schedules a SKIP ordered set transmission once every 1180 to 1538 Symbol times. However, if the transmitter starts a maximum sized TLP transmission right at the 1538 Symbol time boundary when a SKIP ordered set is scheduled to be transmitted, the SKIP ordered set transmission is deferred. Receivers must be able to tolerate SKIP ordered sets that have a maximum separation dependent on the maximum packet payload size a device supports. The formula for the maximum number of Symbols ( $n$ ) between SKIP ordered sets is:

$$n = 1538 + (\text{maximum packet payload size} + 28)$$

The number 28 in the equation is the TLP overhead. It is the largest number of Symbols that would be associated with the header (16 bytes), the optional ECRC (4 bytes), the LCRC (4 bytes), the sequence number (2 bytes) and the framing Symbols STP and END (2 bytes).

## Lane-to-Lane Skew

### Flight Time Will Vary Between Lanes

For wide links, skew between lanes is an issue that can't be avoided and which must be compensated at the receiver. Symbols are sent simultaneously on all lanes using the same transmit clock, but they can't be expected to arrive at the receiver at precisely the same time. Sources of Lane-to-Lane skew include:

- Differences between electrical drivers and receivers
- Printed wiring board impedance variations
- Trace length mismatches

When the serial bit streams carrying a packet arrive at the receiver, this Lane-to-Lane skew must be removed to receive the bytes in the correct order. This process is referred to as de-skewing the link.

### Ordered sets Help De-Skewing

The unique structure of the ordered sets and the fact that they are sent simultaneously on all the lanes makes them useful for detecting timing misalignment between Lanes. The spec doesn't define a method for doing this but in Gen1 and Gen2 the receiver logic can simply look for the COM character on each lane. If it doesn't appear at the same time on all Lanes, then the early arriving COMs are delayed until they all match up on all Lanes.

### Receiver Lane-to-Lane De-Skew Capability

This could be done by adjusting an analog delay line on the incoming signals. Alternatively, it could be done after the elastic buffer, which has the advantage that the arrival time differences are now digitized to Symbol times by the local clock of the receiver (see Figure 11-23 on page 399). If the input to one lane makes it on a clock edge and another one doesn't, the early arrival COMs can simply be delayed by the appropriate number of Symbol clocks to line it up with the late arriving COMs. The fact that the maximum allowable skew at the receiver is a multiple of the clock periods infers that the spec writers probably had an implementation like this in mind (see Table 11-3 on page 399).

# Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

Table 11-3: Allowable Receiver Signal Skew

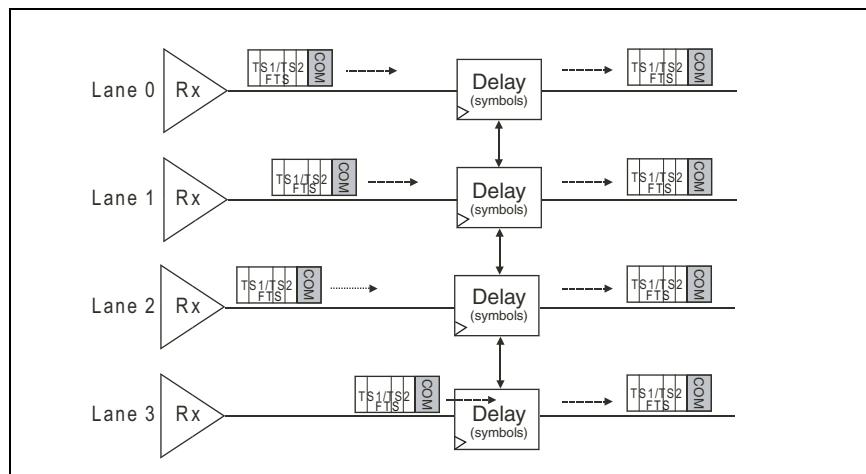
Spec Version	Allowable Rx Skew
Gen1	20 ns (5 clocks at 4ns per Symbol)
Gen2	8 ns (4 clocks at 2ns per Symbol)
Gen3	6 ns (4 clocks at 1.25ns per Symbol)

In Gen3 mode there aren't any COM characters to use for de-skewing, but detecting Ordered Sets can still provide the necessary timing alignment.

## De-Skew Opportunities

An unambiguous pattern is needed on all lanes at the same time to perform de-skewing and any ordered set will do. Link training sends these, but the SKIP ordered set is sent regularly during normal Link operation. Checking its arrival time allows the skew to be checked on an ongoing basis in case it might change based on temperature or voltage. If it does, the Link will need to transition to the Recovery LTSSM state to correct it. If that happens while packets are in flight, however, a receiver error may occur and a packet could be dropped, possibly resulting in replayed TLPs.

Figure 11-23: Receiver's Link De-Skew Logic



## 8b/10b Decoder

### General

The first two generations of PCIe use 8b/10b, while Gen3 does not. Let's explore the operation of it first and then consider the difference for Gen3. Refer to Figure 11-24 on page 401. Each receiver Lane incorporates a 10b/8b decoder which is fed from the Elastic Buffer. The decoder is shown with two lookup tables (the D and K tables) to decode the 10-bit Symbol stream into 8-bit characters plus the D/K# signal. The state of the D/K# signal indicates that the received Symbol is a Data (D) character if a match for the received Symbol is found in the D table, or a Control (K) character if a match for the received Symbol is discovered in the K table.

### Disparity Calculator

The decoder sets the disparity value based on the disparity of the first Symbol received. After the first Symbol, once Symbol lock has been achieved and disparity has been initialized, the calculated disparity for each subsequent Symbol's disparity is expected to follow the rules. If it does not, a Receiver Error is reported.

### Code Violation and Disparity Error Detection

**General.** The error detection logic of the 8b/10b decoder detects illegal Symbols in the received Symbol stream. Some error checking is optional in the receiver, but the spec requires that these errors be checked and reported as a Receiver Error. The two types of errors detected are:

#### Code Violations.

- Any 6-bit sub-block containing more than four 1s or four 0s is in error.
- Any 4-bit sub-block containing more than three 1s or three 0s is in error.
- Any 10-bit Symbol containing more than six 1s or six 0s is in error.
- Any 10-bit Symbol containing more than five consecutive 1s or five consecutive 0s is in error.
- Any 10-bit Symbol that doesn't decode into an 8-bit character is in error.

#### Disparity Errors.

At the receiver a Symbol cannot have a disparity that doesn't match what it should be for the CRD. If it does, a disparity error is detected. Some disparity errors may not be detectable until the subsequent Symbol is processed

## Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

(see Figure 11-25 on page 401). For example, if two bits in a Symbol flip in error, the error may not be visible and the Symbol may decode into a valid 8-bit character. Such an error won't be detected in the Physical Layer.

Figure 11-24: 8b/10b Decoder per Lane

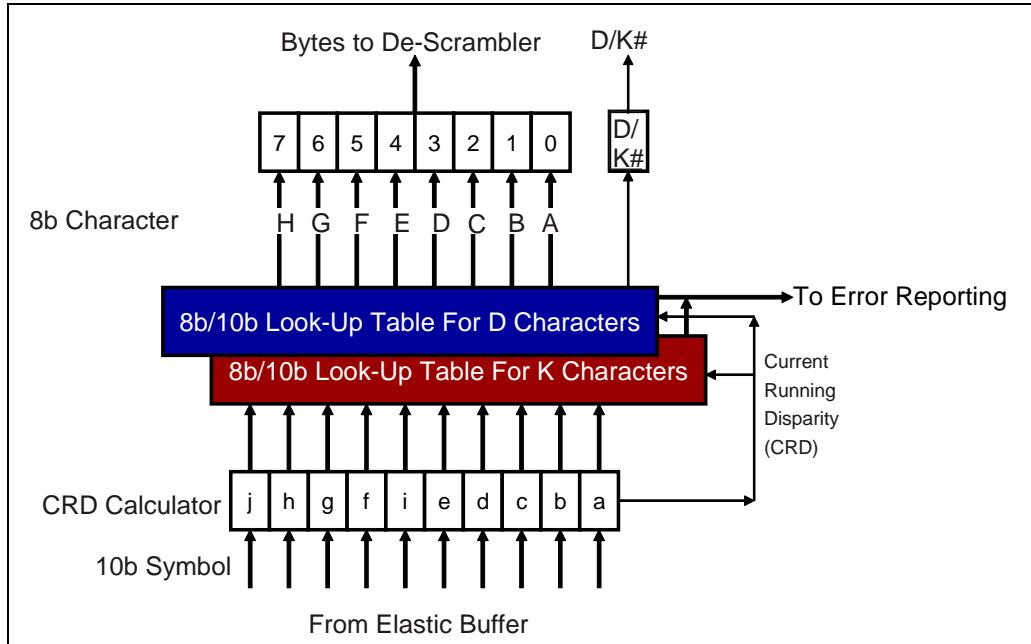


Figure 11-25: Example of Delayed Disparity Error Detection

	CRD	Character	CRD	Character	CRD	Character	CRD
Transmitted Character Stream	-	D21.1	-	D10.2	-	D23.5	+
Transmitted Bit Stream	-	101010 1001	-	010101 0101	-	111010 1010	+
Bit Stream After Error	-	101010 10 <u>1</u> 1	+	010101 0101	+	111010 1010	+
Decoded Character Stream	-	D21.0	+	D10.2	+	Invalid	+

Error occurs here                                  Error detected here

---

## Descrambler

The descrambler is fed by the 8b/10b decoder. It only descrambles Data (D) characters associated with a TLP or DLLP (D/K# is high). It doesn't descramble Control (K) characters or ordered sets because they're not scrambled at the transmitter.

### Some Descrambler Implementation Rules:

- On a multi-Lane Link, descramblers associated with each Lane must operate in concert, maintaining the same simultaneous value in each LFSR.
- Descrambling is applied to 'D' characters associated with TLP and DLLPs including the Logical Idle (00h) sequence. 'D' characters within ordered set are not descrambled.
- 'K' characters and ordered set characters bypass the descrambler logic.
- Compliance Pattern characters are not descrambled.
- When a COM character enters the descrambler, it reinitializes the LFSR value to FFFFh.
- With one exception, the LFSR serially advances eight times for every character (D or K character) received. The LFSR does NOT advance on SKP characters associated with the SKIP ordered sets received. The reason the LFSR is not advanced on detecting SKPs is because there may be a difference between the number of SKP characters transmitted and the SKP characters exiting the Elastic Buffer (as discussed in "Receiver Clock Compensation Logic" on page 396).

### Disabling Descrambling

By default, descrambling is always enabled, but the spec allows it to be disabled for test and debug purposes although no standard software method is given for disabling it. If the descrambler receives at least two TS1/TS2 ordered sets with the "disable scrambling" bit set on all of its configured Lanes, it disables the descrambler.

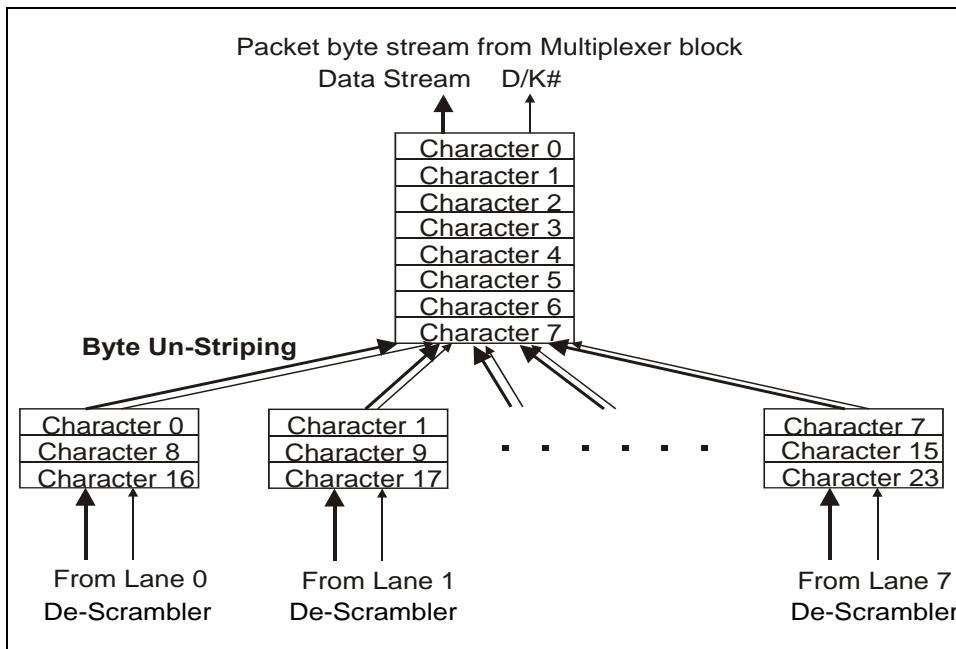
---

## Byte Un-Striping

Figure 11-26 on page 403 shows eight character streams from the descramblers of a x8 Link being un-striped into a single byte stream which is fed to the character filter logic.

## Chapter 11: Physical Layer - Logical (Gen1 and Gen2)

Figure 11-26: Example of x8 Byte Un-Striping



### Filter and Packet Alignment Check

The serial byte stream supplied by the byte un-stripping logic contains TLPs, DLLPs, Logical Idle sequences, Control characters such as STP, SDP, END, EDB, and PADs, as well as the ordered sets. Of these, the Logical Idle sequence, the control characters and ordered sets are detected and eliminated before they get to the next layer. What remains are the TLPs and DLLPs and these are sent to the Rx Buffer along with an indicator of the start and end of each packet.

### Receive Buffer (Rx Buffer)

The Rx Buffer holds received TLPs and DLLPs after the start and end characters have been eliminated. The received packets are ready to send to the Data Link Layer. The interface to the Data Link Layer is not described in the spec, so the designer is free to decide details like data bus width. As an example, we can

# PCI Express Technology

---

assume an interface clock of 250MHz and a Gen1 speed on the Link. For that case, the number of bytes in the data bus between these layers would be the same as the number of Lanes supported.

---

## Physical Layer Error Handling

---

### General

Physical Layer errors are reported as Receiver Errors to the Data Link Layer. According to the spec, some errors must be checked and trigger a receiver error, while others are optional.

Required error checking:

- 8b/10b decode errors: disparity error, illegal Symbol

Optional error checking:

- Loss of Symbol lock (see “Achieving Symbol Lock” on page 396)
- Elastic Buffer overflow or underflow
- Lane deskew errors (see “Lane-to-Lane Skew” on page 398)
- Packets inconsistent with format rules

---

## Response of Data Link Layer to Receiver Error

If the Physical Layer indicates a Receiver Error to the Data Link Layer, the Data Link Layer discards the TLP currently being received and frees any storage allocated for the TLP. It then schedules a NAK to go back to the transmitter of the TLP. That causes the transmitter to replay TLPs from the Replay Buffer, which should automatically correct the error. The Data Link Layer may also direct the Physical Layer to initiate Link re-training.

If the PCI Express Extended Advanced Error Capabilities register set is implemented, a Receiver Error sets the Receiver Error Status bit in the Correctable Error Status register. If enabled, the device can send an ERR\_COR (correctable error) message to the Root Complex.

# **Chapter 11: Physical Layer - Logical (Gen1 and Gen2)**

---

## **Active State Power Management**

There are several Link power states that allow power savings under certain conditions. These are L0s, L1, L2, and L3, which represent progressively lower power and also longer recovery time to get the link back to the fully-operation state of L0. The L0s state can only be entered under hardware control, while L1 can be initiated by hardware or software. Since L0s and L1 can be controlled by hardware, they are referred to by the spec as ASPM (Active State Power Management) states. For more on the details of link and device power management see the section "Active State Power Management (ASPM)" on page 735.

## **Link Training and Initialization**

As we've just briefly mentioned in this chapter, the Physical Layer is also responsible for initializing the link after a reset. However, this topic is too big to cover here and is instead covered in Chapter 14, entitled "Link Initialization & Training," on page 505.

## **PCI Express Technology**

---

---

---

# **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.

# PCI Express Technology

---

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
<b>Gen1 Bandwidth (GB /s)</b>	0.5	1	2	4	6	8	16
<b>Gen2 Bandwidth (GB/s)</b>	1	2	4	8	12	16	32
<b>Gen3 Bandwidth (GB/s)</b>	2	4	8	16	24	32	64

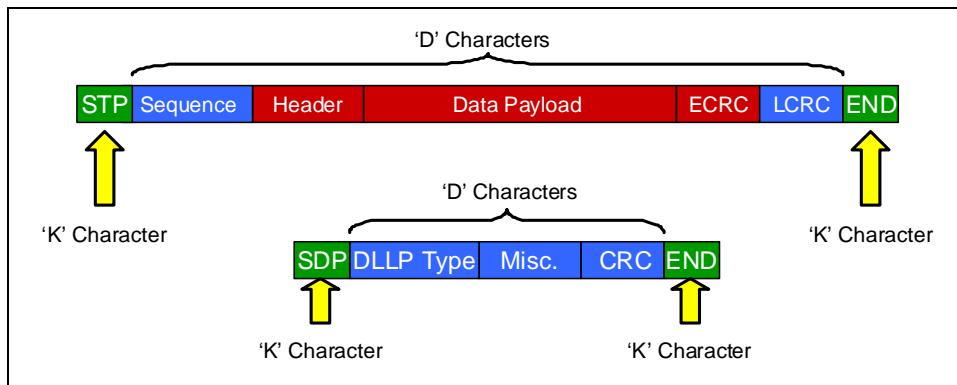
These considerations led to two significant changes to the Gen3 spec compared with the previous generations: a new encoding model and a more sophisticated signal equalization model.

## 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.

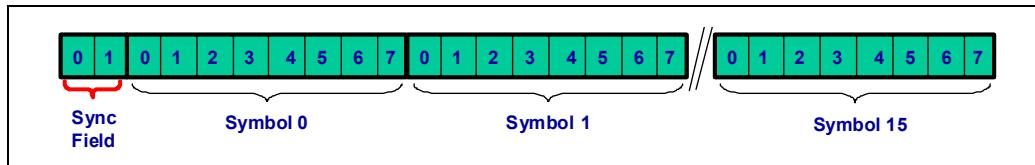
To illustrate the difference between these two encodings, first consider Figure 12-1 that shows the general 8b/10b packet construction. The arrows highlight the Control (K) characters representing the framing Symbols for the 8b/10b packets. Receivers know what to expect by recognizing these control characters. 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.

---

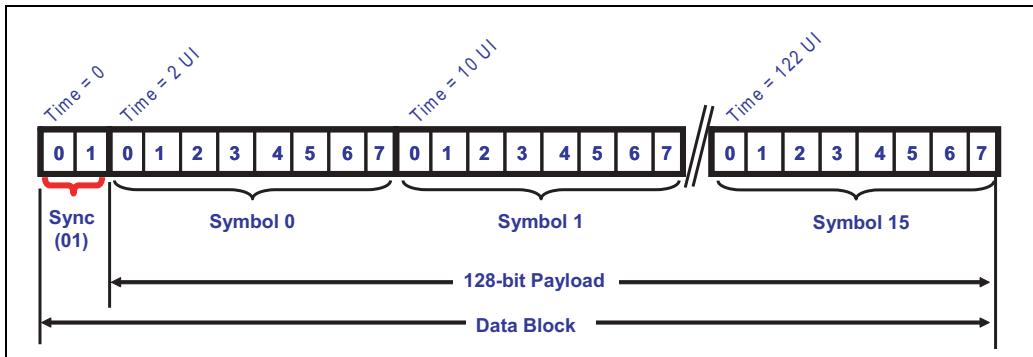
## Lane-Level Encoding

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 followed 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.

## Chapter 12: Physical Layer - Logical (Gen3)

---

Figure 12-3: Sync Header Data Block Example



---

### Block Alignment

Like previous implementations, Gen3 achieves Bit Lock first and then attempts to establish Block Alignment locking. This requires receivers to find the Sync Header that demarcates the Block boundary. Transmitters establish this boundary by sending recognizable EIEOS patterns consisting of alternating bytes of 00h and FFh, as shown in Figure 12-4. Thus, the use of EIEOS has expanded from simply exiting Electrical Idle to also serving as the synchronizing mechanism that establishes Block Alignment. Note that the Sync Header bits immediately precede and follow the EIEOS (not shown in the illustration). See “Achieving Block Alignment” on page 438 for details regarding this process.

Figure 12-4: Gen3 Mode EIEOS Symbol Pattern

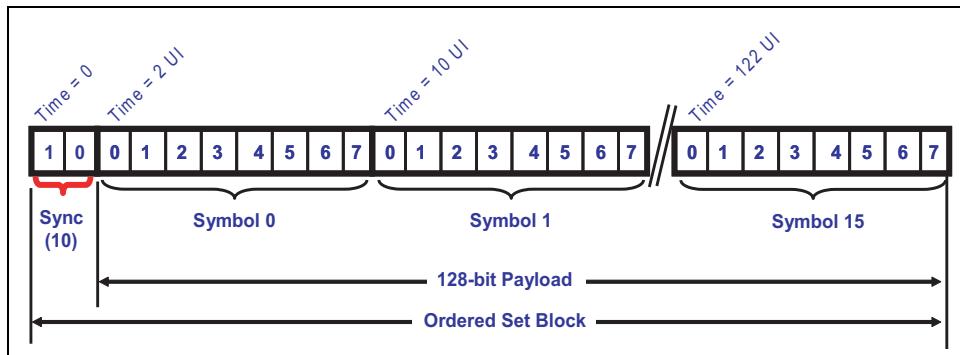
0	00000000
1	11111111
2	00000000
3	11111111
4	00000000
⋮	
13	11111111
14	00000000
15	11111111

## Ordered Set Blocks

Ordered Sets have much the same meaning they did in Gen1 and Gen2. They are used to manage Lane protocol. When an Ordered Set Block is sent it must appear on all the Lanes at the same time and almost always consists of 16 bytes with one exception. The one exception to this size rule is the SOS (SKP Ordered Set) which can have SKP Symbols added or removed in groups of four by clock compensation logic (associated with a Link Repeater for example) and can therefore legally be 8, 12, 16, 20, or 24 bytes long.

The basic format of the Ordered Set Block is similar to the Data Block, except that the Sync Header bits are reversed, as shown in Figure 12-5 on page 412.

Figure 12-5: Gen3 x1 Ordered Set Block Example



The spec defines seven Ordered Sets for Gen3 (one additional Ordered Set over Gen1 and Gen2 PCIe). In most cases, their functionality is the same as it was for the previous generations.

1. SOS - Skip Ordered Set: used for clock compensation. See “Ordered Set Example - SOS” on page 426 for more detail.
2. EIOS - Electrical Idle Ordered Set: used to enter Electrical Idle state
3. EIEOS - Electrical Idle Exit Ordered Set: used for two purposes now:
  - Electrical Idle Exit as before
  - Block alignment indicator for 8.0 GT/s
4. TS1 - Training Sequence 1 Ordered Set
5. TS2 - Training Sequence 2 Ordered Set
6. FTS - Fast Training Sequence Ordered Set
7. SDS - Start of Data Stream Ordered Set: new - see “Data Stream and Data Blocks” on page 413 for more

## Chapter 12: Physical Layer - Logical (Gen3)

To give the reader an example of the Ordered Set structure, Figure 12-6 shows the content of an FTS Ordered Set when running at 8.0 GT/s. An Ordered Set Block is only recognized as an Ordered Set by the Sync Header, and identified as an FTS type by the first Symbol in the Block. The right-hand side of the figure lists the Ordered Set Identifiers (the first Symbol for each Ordered Set) that serve to identify the type of Ordered Set is being transmitted.

Figure 12-6: Gen3 FTS Ordered Set Example

FTS Ordered Set		Ordered Set Identifiers	
Symbol	Value	Ordered Set	First Symbol
Sync Header	01b	EIEOS	00h
0	55h	EIOS	66h
1	47h	FTS	55h
2	4Eh	SDS	E1
3	C7h	TS1	1Eh
4	CCh	TS2	2Dh
5	C6h	SKP	AAh
6	C9h		
7	25h		
8	6Eh		
9	ECh		
10	88h		
11	7Fh		
12	80h		
13	8Dh		
14	8Bh		
15	8Eh		

## Data Stream and Data Blocks

The Link enters a Data Stream by sending an SDS Ordered Set and transitioning to the L0 Link state. While in a Data Stream multiple Data Blocks are transferred, until the Data Stream ends with an EDS Token (unless an error ends it early). An EDS Token always occupies the last four Symbols of the Data Block that precedes an Ordered Set. An exception is made for Skip Ordered Sets because they do not interrupt a Data Stream as long as certain conditions are

# PCI Express Technology

met that are discussed later. A Data Stream is no longer in effect when the Link state transitions out of the L0 state to any other Link state, such as Recovery. For more on Link states, see “Link Training and Status State Machine (LTSSM)” on page 518.

## Data Block Frame Construction

Data Blocks comprise TLPs, DLLP, and Tokens that are used to deliver the information. Five types of Data structures (called Tokens) are also used within a Data Block. Each has patterns for easy detection by the receiver. Three of the token may be sent at the beginning of a block (i.e., immediately following a Sync Data Block). These include:

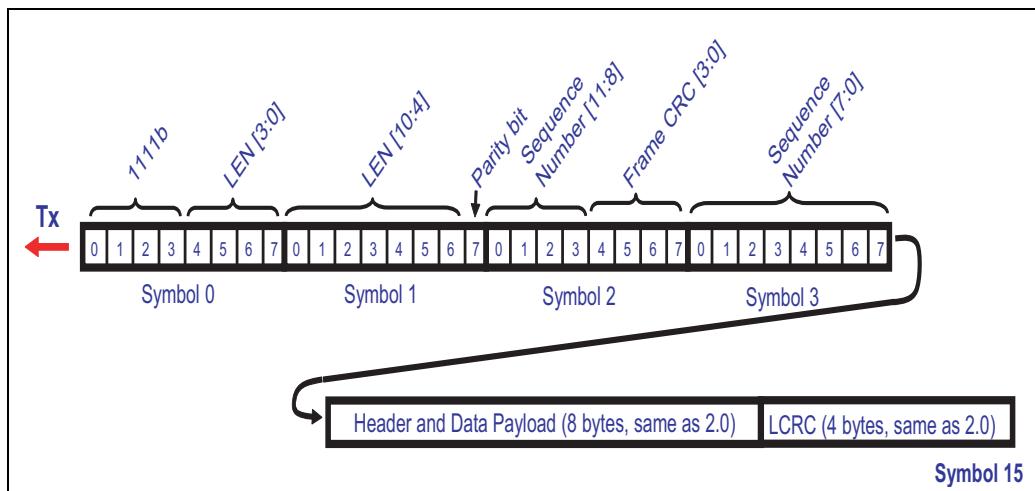
- Start TLP (STP) — followed by a TLP
- Start DLLP (SDP) — followed by a DLLP
- Logical Idle (IDLA) — sent when there is no packet activity

The remaining Tokens are delivered at the end of the Data Block:

- End of Data Stream (EDS) — Precedes the transition to Ordered Sets
- End Bad (EDB) — reports a nullified packet has been detected

Figure 12-7 provides an example of a Data Block consisting of a single lane TLP transmission.

Figure 12-7: Gen3 x1 Frame Construction Example



# **Chapter 12: Physical Layer - Logical (Gen3)**

---

In summary, the contents of a given Data Block vary depending on the activity:

- IDLs — when no packets are being delivered Data Blocks consist of nothing but IDL. (The spec designates IDL as one of the Tokens)
- TLPs — One or more TLPs may be sent in a given Data Block depending on the link width.
- DLLPs — One or more DLLPs may be sent in a Data Block.
- Combinations of the activity listed above may be delivered in a single Data Block

## **Framing Tokens**

The spec defines five Framing Tokens (or just “Tokens” for short) that are allowed to appear in a Data Block, and those are repeated for convenience here in Figure 12-8 on page 417. The five Tokens are:

1. **STP - Start TLP:** Much like earlier version, but now includes dword count for the entire packet.
2. **SDP - Start DLLP**
3. **EDB - End Bad:** Used to nullify a TLP the way it was in earlier Gen1 and Gen2 designs, but now four EDB symbols in a row are sent. The END (End Good) symbol is done away now; if not explicitly marked as bad, the TLP will be assumed to be good.
4. **EDS - End of Data Stream:** Last dword of a Data Stream, indicating that at least one Ordered Set will follow. Curiously, the Data Stream may not actually be ended by this event. If the Ordered Set that follows it is an SOS and is immediately followed by another Data Block, the Data Stream continues. If the Ordered Set that follows the EDS is anything other than SOS, or if the SOS is not followed by a Data Block, the Data Stream ends.
5. **IDL - Logical Idle:** The Idle Token is simply data zero bytes sent during Link Logical Idle state when no TLPs or DLLPs are ready to transmit.

The difference between the way the spec shows the Tokens and the way they’re presented in Figure 12-8 on page 417 is that this drawing shows both bytes and bits in little-endian order instead of the big-endian bit representation used in the spec. The reason it’s shown that way is to illustrate the order that the bits will actually appear on the Lane.

## **Packets**

The STP and SDP, indicate the start of a packet as shown in Figure 12-7

- **TLPs.** An **STP** Token consists of a nibble of 1's followed by an 11-bit dword-length field. The length counts all the dwords of the TLP, including the

# PCI Express Technology

---

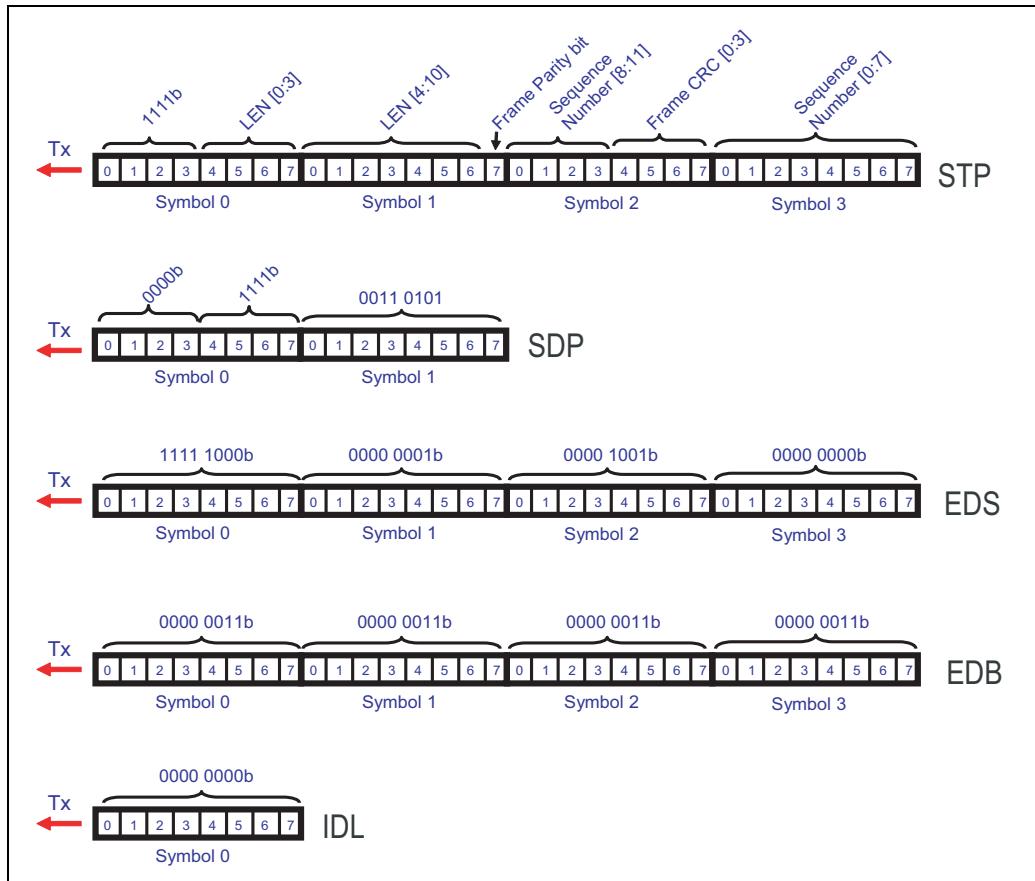
Token, header, optional data payload, optional digest, and LCRC. That allows the receiver to count dwords to recognize where the TLP ends. Consequently, it's very important to verify that the Length field doesn't have an error, and so it has a 4-bit Frame CRC, and an even parity bit that protects both the Length and Frame CRC fields. The combination of these bits provides a robust triple-bit-flip detection capability for the Token (as many as 3 bits could be incorrect and it would still be recognized as an error). The 11-bit Length field allows for a TLP of 2K dwords (8KB) for the entire TLP.

- **DLLPs.** The **SDP** Token indicates the beginning of a DLLP and doesn't include a length field because it will always be exactly 8 bytes long: the 2-byte Token is followed by 4 bytes of DLLP payload and 2 bytes of DLLP LCRC. Perhaps coincidentally, this DLLP length is the same as it was in earlier PCIe generations, but they also do not have an end good symbol.

The **EDB** Token is added to the end of TLPs that are nullified. For a normal TLP, there is no "end good" indication; it's assumed to be good unless explicitly marked as bad. If the TLP ends up being nullified, the LCRC value is inverted and an EDB Token is appended as an extension of the TLP, although it's not included in the length value. Physical layer receivers must check for the EDB at the end of every TLP and inform the Link layer if they see one. Not surprisingly, receiving an EDB at any time other than immediately after a TLP will be considered to be a Framing Error.

## Chapter 12: Physical Layer - Logical (Gen3)

Figure 12-8: Gen3 Frame Token Examples



### Transmitter Framing Requirements

To begin this discussion, it will be helpful first to define a couple of things. First, recall that a Data Stream starts with the first Symbol following an SDS and it may contain Data Blocks made up of Tokens, TLPs and DLLPs. The Data Stream finishes with the last Symbol before an Ordered Set other than SOS, or when a Framing Error is detected. During a Data Stream no Ordered Sets can be sent except for the SOS.

Secondly, since framing problems will usually result in a Framing Error, it will help to explain what happens in that case. When Framing Errors occur, they are

# PCI Express Technology

---

considered Receiver Errors and will be reported as such. The Receiver stops processing the Data Stream in progress and will only process a new Data Stream when it sees an SDS Ordered Set. In response to the error, a recovery process is initiated by directing the LTSSM to the Recovery state from L0. The expectation is that this will be resolved in the Physical Layer and will not require any action by the upper layers. In addition, the spec states that the round-trip time to accomplish this is expected to take less than 1µs from the time both Ports have entered Recovery.

Now, with that background in place, let's continue with the framing requirements. While in a Data Stream, a transmitter must observe the following rules:

- When sending a TLP:
  - An STP Token must be immediately followed by the entire contents of the TLP as delivered from the Link Layer, even if it's nullified.
  - If the TLP was nullified, the EDB Token must appear immediately after the last dword of the TLP, but must not be included in the TLP length value.
  - An STP cannot be sent more than once per Symbol Time on the Link.
- When sending a DLLP:
  - An SDP Token must be immediately followed by the entire contents of the DLLP as delivered from the Data Link Layer.
  - An SDP cannot be sent more than once per Symbol Time on the Link.
- When sending an SOS (SKP Ordered Set) within a Data Stream:
  - Send an EDS Token in the last dword of the current Data Block.
  - Send the SOS as the next Ordered Set Block.
  - Send another Data Block immediately after the SOS. The Data Stream resumes with the first Symbol of this subsequent Data Block.
  - If multiple SOS's are scheduled, they can't be back-to-back as they were in earlier generations. Instead, each one must be preceded by a Data Block that ends with the EDS Token. The Data block can be filled with TLPs, DLLPs or IDLs during this time.
- To end a Data Stream, send the EDS Token in the last dword of the current Data Block and follow that with either the EIOS to go into a low power Link state, or an EIEOS for all other cases.
- The IDL Token must be sent on all Lanes if a TLP, DLLP, or other Framing Token is not being sent on the Link.
- For multi-Lane Links:
  - After sending an IDL Token, the first Symbol of the next TLP or DLLP must be in Lane 0 when it starts. An EDS Token must always be the last dword of a Data Block and therefore may not always follow that rule.
  - IDL Tokens must be used to fill in dwords during a Symbol Time that would otherwise be empty. For example, if a x8 Link has a TLP that

# **Chapter 12: Physical Layer - Logical (Gen3)**

---

ends in Lane 3, but the sender doesn't have another TLP or a DLLP ready to start in Lane 4, then IDLs must fill in the remaining bytes until the end of that Symbol Time.

- Since packets are still multiples of 4 bytes as they were in the earlier generations, they'll start and end on 4-Lane boundaries. For example, a x8 Link with a DLLP that ends in Lane 3 could start the next TLP by placing its STP Token in Lane 4.

## **Receiver Framing Requirements**

When a Data Stream is seen at the Receiver, the following rules apply:

- When Framing Tokens are expected, Symbols that look like anything else will be Framing Errors.
- Some error checks and reports shown in the list below are optional, and the spec points out that they are independently optional.
- When an STP is received:
  - Receivers must check the Frame CRC and Frame Parity fields, and any mismatch will be a Framing Error. (Note that an STP Token with a Framing Error isn't considered to be part of a TLP when reporting this error.).
  - The Symbol immediately after the last DW of the TLP is the next Token to process, and Receivers must check to see whether it's the start of an EDB Token showing that the TLP has been nullified.
  - Optionally check for length value of zero; if detected, it's a Framing Error.
  - Optionally check for the arrival of more than one STP Token in the same Symbol Time. If checking and detected, this is a Framing Error.
- When an EDB is received:
  - Receiver must inform the Link Layer as soon as the first EDB Symbol is detected, or after any of the remaining bytes of it have been received.
  - If any Symbols in the Token are not EDBs, the result is a Framing Error.
  - The only legal time for an EDB Token is right after a TLP; any other use will be a Framing Error.
  - The Symbol immediately following the EDB Token will be the first Symbol of the next Token to be processed.
- When an EDS Token is received as the last DW of a Data Block:
  - Receivers must stop processing the Data Stream.
  - Only a SKP, EIOS, or EIEOS Ordered Set will be acceptable next; receiving any other Ordered set will be a Framing Error.
  - If a SKP Ordered Set is received after an EDS, Receivers must resume Data Stream processing with the first Symbol of the Data Block that follows, unless a Framing Error was detected.

- When an SDP Token is received:
  - The Symbol immediately after the DLLP is the next Token to be processed.
  - Optionally check for more than one SDP Token in the same Symbol Time. If checking and this occurs, it is a Framing Error.
- When an IDL Token is received:
  - The next Token is allowed to begin on any DW-aligned Lane following the IDL Token. For Links that are x4 or narrower, that means the next Token can only start in Lane 0 of the next Symbol Time. For wider Links there are more options. For example, a x16 Link could start the next Token in Lane 0, 4, 8, or 12 of the current Symbol Time.
  - The only Token that would be expected in the same Symbol Time as an IDL would be another IDL or an EDS.
- While processing a Data Stream, Receivers will see the following as Framing Errors:
  - An Ordered Set immediately following an SDS.
  - A Block with an illegal Sync Header (11b or 00b). This can optionally be reported in the Lane Error Status register.
  - An Ordered Set Block on any Lane without receiving an EDS Token in the previous Block.
  - A Data Block immediately following an EDS Token in the previous block.
  - Optionally, verify that all Lanes receive the same Ordered Set.

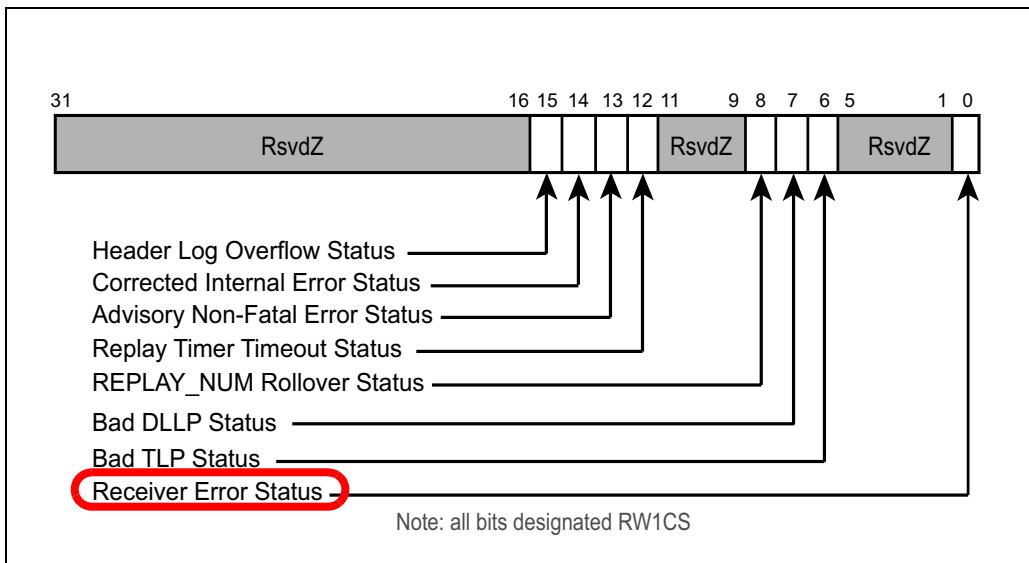
## Recovery from Framing Errors

If a Framing Error is seen while processing a Data Stream, the Receiver must:

- Report a Receiver Error (if the optional Advanced Error Reporting registers are available, set the status bit shown in Figure 12-9 on page 421).
- Stop processing the Data Stream. Processing a new Data Stream can begin when the next SDS Ordered Set is seen.
- Initiate the error recovery process. If the Link is in the L0 state, that will involve a transition to the Recovery state. The spec says that the time through the Recovery state is “expected” to be less than 1μs.
- Note that recovery from Framing Errors is not necessarily expected to directly cause Data Link Layer initiated recovery activity via the Ack/Nak mechanism. Of course, if a TLP is lost or corrupted as a result of the error, then a replay event will be needed.

## Chapter 12: Physical Layer - Logical (Gen3)

Figure 12-9: AER Correctable Error Register



## Gen3 Physical Layer Transmit Logic

Figure 12-10 on page 422 illustrates a conceptual block diagram of the Physical Layer transmit logic that supports Gen3 speeds. The overall design is very similar to Gen2 so there's no need to go through all the details again but there are some differences. Those who are new to PCIe are encouraged to review the earlier chapter called "Physical Layer - Logical (Gen1 and Gen2)" on page 361 to learn the basics of the Physical Layer design. Let's start at the top of the diagram and explain the changes for Gen3 along the way. As before, it's important to point out that this implementation is only for instructional purposes and is not meant to show an actual Gen3 Physical Layer implementation.

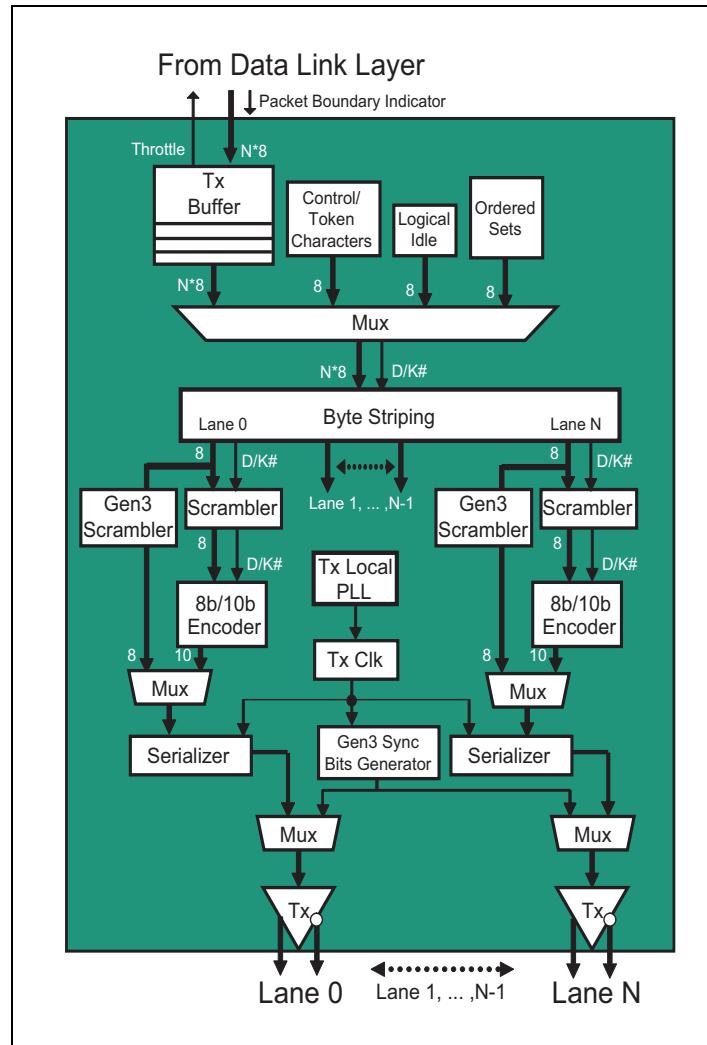
### Multiplexer

TLPs and DLLPs arrive from the Data Link Layer at the top. The multiplexer mixes in the STP or SDP Tokens necessary to build a complete TLP or DLLP. The previous section described the Token formats.

# PCI Express Technology

---

Figure 12-10: Gen3 Physical Layer Transmitter Details



## **Chapter 12: Physical Layer - Logical (Gen3)**

---

Gen3 TLP boundaries are defined by the dword count in the Length field of the STP Token at the beginning of a TLP packet, therefore, no END frame character is needed.

When ending a Data Stream or just before sending an SOS, the EDS Token is muxed into the Data Stream. At regular intervals, based on a Skip timer, an SOS is inserted into the Data Stream by the multiplexer. Other Ordered-Sets such as TS1, TS2, FTS, EIEOS, EIOS, SDS may also be muxed based on Link requirements and are outside the Data Stream.

Packets are transmitted in Blocks which are identified by the 2-bit Sync Header. The Sync Header is added by the multiplexer. However, the Sync Header is replicated on all Lanes of a multi-Lane Link by the Byte Striping logic.

When there are no packets or Ordered Sets to send but the Link is to remain active in L0 state, the IDL (Logical Idle, or data zero) Tokens are used as fillers. These are scrambled just like other data bytes and are recognized as filler by the Receiver.

---

### **Byte Striping**

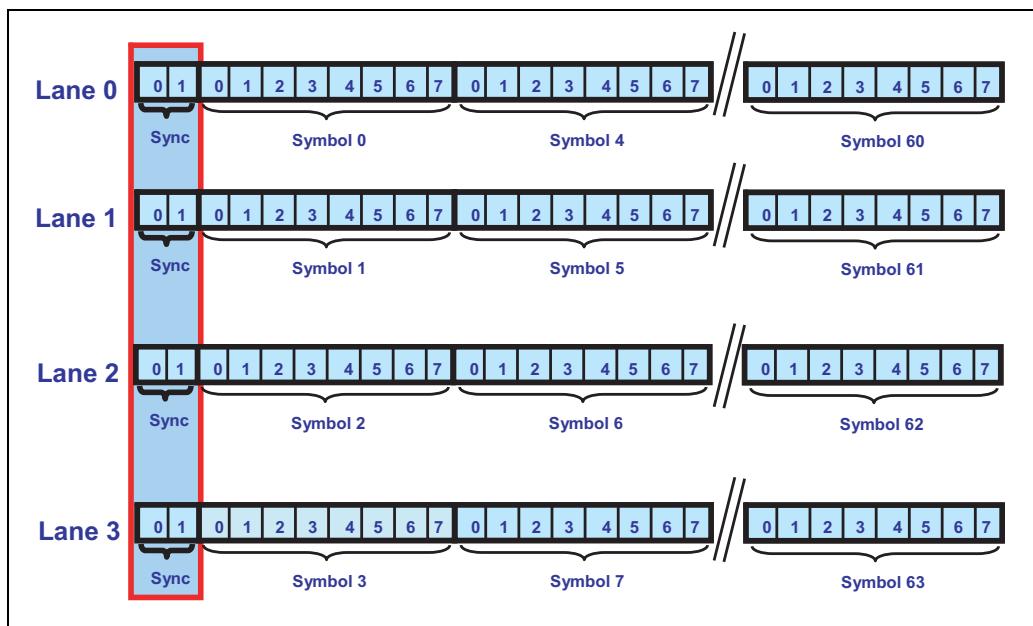
This logic spreads the bytes to be delivered across all the available Lanes. The framing rules were described earlier in “Transmitter Framing Requirements” on page 417, so now let’s look at some examples and discuss how the rules apply.

Consider first the example shown in Figure 12-11 on page 424, where a 4-Lane Link is illustrated. Notice that the Sync Header bits appear on all the Lanes at the same time when a new Block begins and define the block type (a Data Block in this example). Block encoding is handled independently for each Lane, but the bytes (or symbols) are striped across all the Lanes just as they were for the earlier generations of PCIe.

# PCI Express Technology

---

Figure 12-11: Gen3 Byte Striping x4



## Byte Striping x8 Example

Next, consider the x8 Link shown in Figure 12-12 on page 425, which is an example from the spec redrawn to make it easier to read. Here the bit stream is vertical instead of horizontal. At the top we can see that the Sync bits, shown in little-endian order as required, appear on all Lanes simultaneously and indicate that a Data Block is starting.

In this example, a TLP is sent first, so Symbols 0 - 4 contain the STP framing Token, which includes a length of 7 DW for the entire TLP including the Token. The receiver needs to know the length of the TLP because for 8 GT/s speeds there is no END control character. Instead, the receiver counts the dwords and if there is no EDB (End Bad) observed, the TLP is assumed to be good. In this case, the TLP ends on Lane 3 of Symbol 3.

## Chapter 12: Physical Layer - Logical (Gen3)

*Figure 12-12: Gen3 x8 Example: TLP Straddles Block Boundary*

	Lane 0	Lane 1	Lane 2	Lane 3	Lane 4	Lane 5	Lane 6	Lane 7
Sync	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
Symbol 0	STP Token: Length=7, CRC, Parity, Seq Num							
Symbol 1			TLP					
Symbol 2								
Symbol 3	LCRC				SDP Token			
Symbol 4	DLLP				IDL	IDL	IDL	IDL
Symbol 5	IDL	IDL	IDL	IDL	IDL	IDL	IDL	IDL
Symbol 6	STP Token: Length=23, CRC, Parity, Seq Num					DW 1		
Symbol 7	DW 2					DW 3		
Symbol 15	DW 18					DW 19		
Sync	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
Symbol 0	DW 20					DW 21		
Symbol 1	LCRC				IDL	IDL	IDL	IDL

Next a DLLP is sent beginning with the SDP Token on Lanes 4 and 5. Since a DLLP is always 8 Symbols long, it will finish in Lane 3 of Symbol 4. Momentarily, there are no other packets to send, so IDL Symbols are transferred until another packet is ready. When IDLs are sent, the next STP Token can only start in Lane 0. In the example, the TLP starts in Lane 0 of Symbol 6.

The packet length for the next TLP is 23 DW and that presents an interesting situation because there are only 20 dwords available before the next Block boundary. When the Data Block ends the transmitter sends Sync and continues TLP transmission during Symbol 0 of the next Block. In other words, Packets simply straddle Block boundaries when necessary. Finally, the TLP finishes in Lane 3 of Symbol 1. Once again there are no packets ready to send, so IDLs are sent.

### Nullified Packet x8 Example

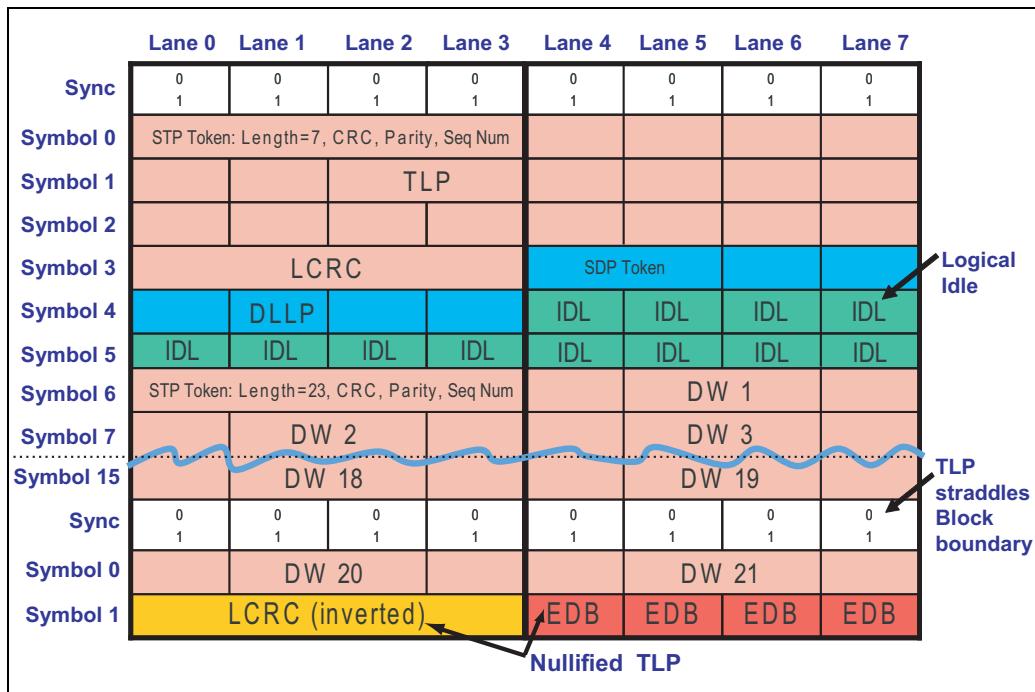
Nullified TLPs can occur when a TLP is being transferred across a switch to reduce latency. This is called Switch Cut-Through operation. The reader may choose to review the section entitled "Switch Cut-Through Mode" on page 354 before proceeding with this discussion.

# PCI Express Technology

A nullified TLP can occur when a switch forwards a packet to the egress port before having received the packet at the ingress port and before error checking. Because an error was detected in this example, the TLP must be nullified.

Figure 12-13 illustrates the steps taken to nullify TLP. The TLP being sent by the egress port, starts in the first block (Lane 0 of Symbol 6). When the error is detected, the egress port inverts the CRC (Lanes 0-3 of Symbol 1) and adds an EDB token immediately following the TLP (Lanes 4-7 of symbol 1). Together, those two changes make it clear to the Receiver that this TLP has been nullified and should be discarded. Note that the EDB bytes are not included in the packet length field, because they dynamically added to a packet in flight when an error occurs.

Figure 12-13: Gen3 x8 Nullified Packet



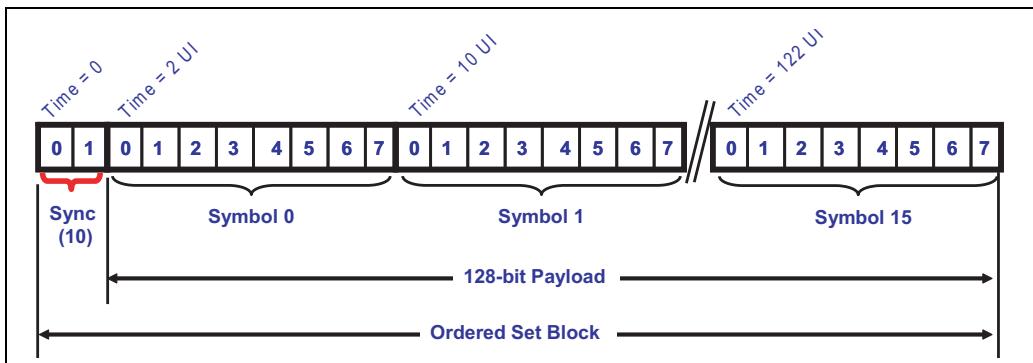
## Ordered Set Example - SOS

Now let's consider an example of Ordered Set transmission. As shown in Figure 12-14 on page 427, an Ordered Set is indicated by the 2-bit Sync Header value of 01b. The bytes that follow will be understood by the receiver to make up an Ordered Set that is always 16 bytes (128 bits) in length. The one exception is the

## Chapter 12: Physical Layer - Logical (Gen3)

SOS (Skip Ordered Set), because it can be changed by intermediate receivers in increments of 4 bytes at a time for clock compensation. Consequently, an SOS is legally allowed to be 8, 12, 16, 20, or 24 Symbols in length. In the absence of a Link repeater device that does not add or delete SKPs in a SOS, a SOS will also be made up of 16 bytes.

Figure 12-14: Gen3 x1 Ordered Set Construction



To illustrate an Ordered Set, let's use an SOS to show the various features and how they work together. Consider Figure 12-15 on page 428, where a Data Block is followed by an SOS. The framing rules state that the previous Data Block must end with an EDS Token in the last dword to let the receiver know that an Ordered Set is coming. If the current Data Stream is to continue, the Ordered Set that follows must be an SOS, and that must be followed in turn by another Data Block. This example doesn't show it, but it's possible that a TLP might be incomplete at this point and would straddle the SOS by resuming transmission in the Data Block that must immediately follow the SOS.

Receiving the EDS Token means that the Data Stream is either ending or pausing to insert an SOS. An EDS is the only Token that can start on a dword-aligned Lane in the same Symbol Time as an IDL, and this example does just that, beginning in Lane 4 of Symbol Time 15. Recall that EDS must also be in the last dword of the Data Block. According to the receiver framing requirements, only an Ordered Set Block is allowed after an EDS and must be an SOS, EIOS, or EIEOS or else it will be seen as a framing error. As was true for earlier spec versions, the Ordered Sets must appear on all Lanes at the same time. Receivers may optionally check to ensure that each Lane sees the same Ordered Set.

In our example, a 16 byte SOS is seen next, and is recognized by the Ordered Set Sych Header as well as the SKP byte pattern. There are always 4 Symbols at the end of the SOS that contain the current 24-bit scrambler LFSR state. In Symbol

# PCI Express Technology

---

12 the Receiver knows that the SKP characters have ended and also that the Block has three more bytes to deliver per Lane. These are the output of the scrambling logic LFSR, as shown in Table 12-2 on page 428.

Figure 12-15: Gen3 x8 Skip Ordered Set (SOS) Example

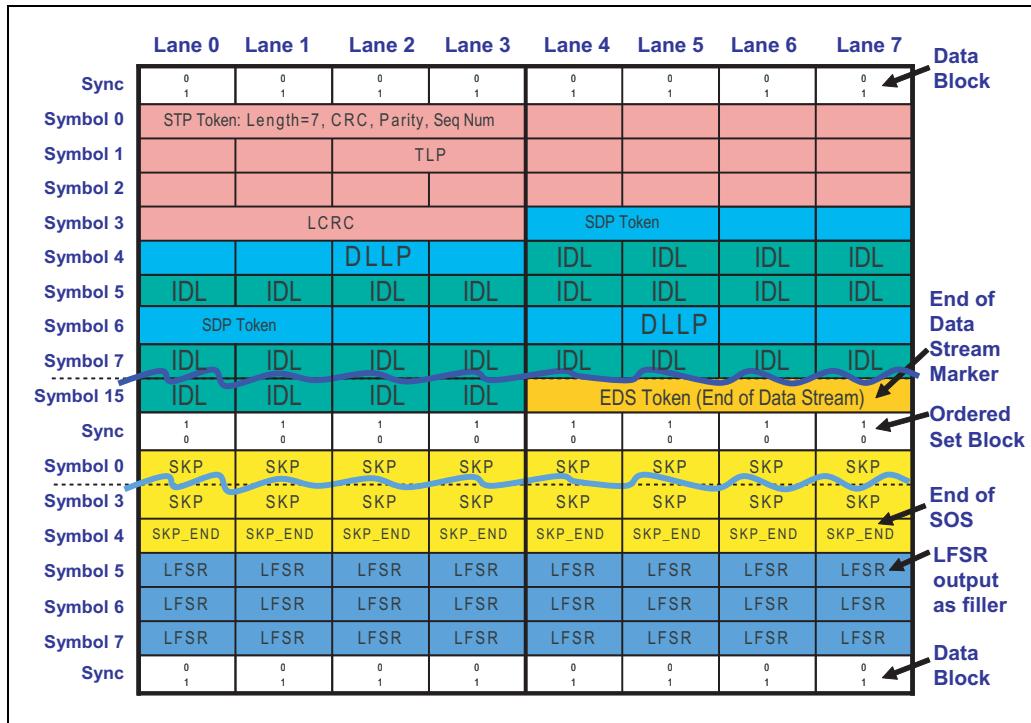


Table 12-2: Gen3 16-bit Skip Ordered Set Encoding

Symbol Number	Value	Description
0 to 11	AAh	SKP Symbol. Since Symbol 0 is the Ordered Set Identifier, this is seen as an SOS.
12	E1h	SKP-END Symbol, which indicates that the SOS will be complete after 3 more Symbols

## Chapter 12: Physical Layer - Logical (Gen3)

---

Table 12-2: Gen3 16-bit Skip Ordered Set Encoding (Continued)

Symbol Number	Value	Description
13	00-FFh	a) If LTSSM state is Polling.Compliance: AAh b) Else if prior block was a Data Block: Bit [7] = Data Parity Bit [6:0] = LFSR [22:16] c) Else Bit [7] = ~LFSR [22] Bit [6:0] = LFSR [22:16]
14	00-FFh	a) If LTSSM state is Polling.Compliance: Error_Status [7:0] b) Else LFSR [15:8]
15	00-FFh	a) If LTSSM state is Polling.Compliance: Error_Status [7:0] b) Else LFSR [7:0]

The Data Parity bit mentioned in the table is the even parity of all the Data Block scrambled bytes that have been sent since the most recent SDS or SOS and is created independently for each Lane. Receivers are required to calculate and check the parity. If the bits don't match, the Lane Error Status register bit corresponding to the Lane that saw the error must be set, but this is not considered a Receiver Error and does not initiate Link retraining.

The 8-bit Error\_Status field only has meaning when the LTSSM is in the Polling.Compliance state (see "Polling.Compliance" on page 529 for more details). For our example of an SOS following a Data Block, byte 13 is the Data Parity bit and LFSR[22:16], while the last two bytes are LFSR bits [15:0].

### Transmitter SOS Rules

The SOS rules for Transmitters when using 128b/130b include:

- An SOS must be scheduled to occur within 370 to 375 blocks. In Loopback mode, however, the Loopback Master must schedule two SOS's within that time, and they must be no more than two blocks from each other.
- SOS's can still only be sent on packet boundaries and may be accumulated as a result. However, consecutive SOS's are not permitted; they must be separated by a Data Block.
- It's recommended that SOS timers and counters be reset whenever the Transmitter is Electrically Idle.

# PCI Express Technology

---

- The Compliance SOS bit in Link Control Register 2 has no effect when using 128b/130b. (It's used to disable SOSs during Compliance testing for 8b/10b, but that isn't an option for 128b/130b.)

## Receiver SOS Rules

The Skip Ordered Set rules for Receivers when using 128b/130b include:

- They must tolerate receiving SOS's at an average interval of 370-375 blocks. Note that the first SOS after Electrical Idle may arrive earlier than that, since Transmitters are not required to reset SOS timers during Electrical Idle time.
- Receivers must check to see that every SOS in a Data Stream is preceded by a Data Block that ends with EDS.

---

## Scrambling

The scrambling logic for 128b/130b is modified from the previous PCIe generations to address the two issues that 8b/10b encoding handled automatically: maintaining DC Balance and providing a sufficient transition density. By way of review, recall that DC Balance means the bit stream has an equal number of ones and zeros. This is intended to avoid the problem of "DC wonder", in which the transmission medium is charged toward one voltage or the other so much, by a prevalence of ones or zeros, that it becomes difficult to switch the signal within the necessary time. The other problem is that clock recovery at the Receiver needs to see enough edges in the input signal to be able to compare them to the recovered clock and adjust the timing and phase as needed.

Without 8b/10b to handle these issues, three steps were taken: First, the new scrambling method improves both transition density and DC Balance over longer time periods, but doesn't guarantee them over short periods the way 8b/10b did. Second, the TS1 and TS2 Ordered Set patterns used during training include fields that are adjusted as needed to improve DC Balance. And third, Receivers must be more robust and tolerant of these issues than they were in the earlier generations.

## Number of LFSRs

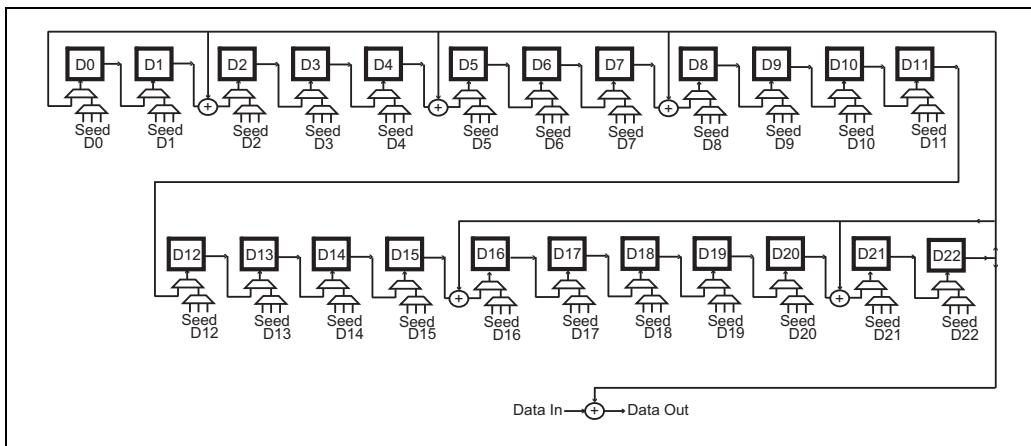
At the lower data rates every Lane was scrambled in the same way, so a single Linear-Feedback Shift Register (LFSR) could supply the scrambling input for all of them. For Gen3, though, the designers wanted different scrambling values for neighboring Lanes. The reasons probably include a desire to decrease the possibility of cross-talk between the Lanes by scrambling their outputs with respect to each other and avoid having the same value on each Lane, as might

## Chapter 12: Physical Layer - Logical (Gen3)

happen when sending IDLs. The spec describes two approaches to achieving this goal, one that emphasizes lower latency and one that emphasizes lower cost.

**First Option: Multiple LFSRs.** One solution is to implement a separate LFSR for each Lane, and initialize each with a different starting value or “seed”. This has the advantage of simplicity and speed, at the cost of adding logic. As shown in Figure 12-16, each LFSR creates a pseudo-random output based on the polynomial given in the spec as  $G(X) = X^{23} + X^{21} + X^{16} + X^8 + X^5 + X^2 + 1$ . This polynomial is longer than the previous version and also behaves a little differently because of the different seed values. Eight different seed values for each Lane are specified requiring eight different LFSRs, one per Lane 0 through 7.

Figure 12-16: Gen3 Per-Lane LFSR Scrambling Logic



The 24-bit seed value for each Lane is listed in Table 12-3 on page 432. The series repeats itself, meaning the seed for Lane 8 will be the same as Lane 0, so only the first 8 values are shown. Every Lane uses the same LFSR and the same tap points to create the scrambling output, and the different seed values give the desired difference.

Table 12-3: Gen3 Scrambler Seed Values

Lane	Seed Value
0	1DBFBCh
1	0607BBh
2	1EC760h
3	18C0DBh
4	010F12h
5	19CFC9h
6	0277CEh
7	1BB807h

**Second Option: Single LFSR.** The alternative solution, illustrated in Figure 12-17 on page 433 for Lanes 2, 10, 18, and 26, is to use just one LFSR and create the scrambling inputs for each Lane by XORing different tap points together. Since there's only one LFSR, the seed value is the same for all Lanes (all ones), but the scrambling "Tap Equation" for each Lane is derived by combining different tap points, as shown in Table 12-4 on page 433. The spec also notes that 4 of the Lanes Tap Equations can be derived by XORing the tap values of their bit neighbors:

- Lane 0 = Lane 7 XOR Lane 1 (note that the process of going to lower Lane numbers wraps around, with the result that Lane 7 is considered lower than Lane 0)
- Lane 2 = Lane 1 XOR Lane 3
- Lane 4 = Lane 3 XOR Lane 5
- Lane 6 = Lane 5 XOR Lane 7

The single-LFSR solution uses fewer gates than the multi-LFSR version does, but incurs extra latency through the XOR process, providing a different cost/performance option.

## Chapter 12: Physical Layer - Logical (Gen3)

Figure 12-17: Gen3 Single-LFSR Scrambler

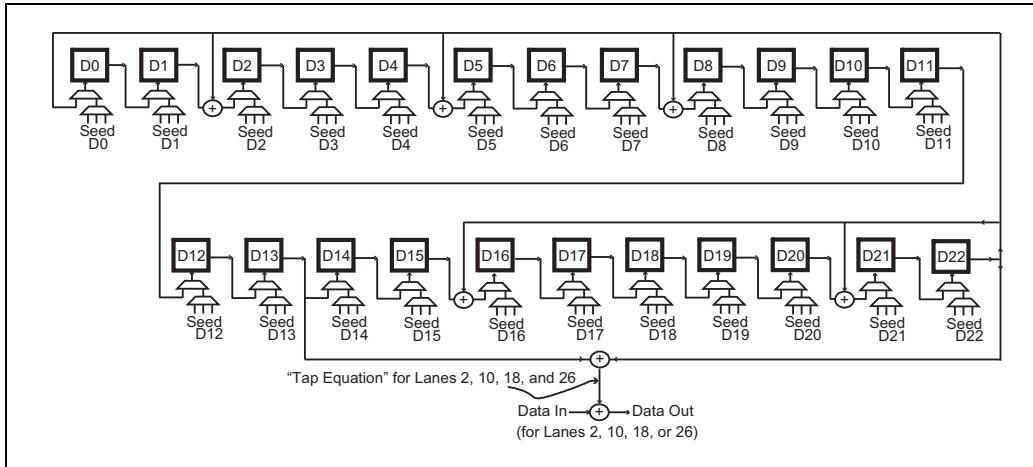


Table 12-4: Gen3 Tap Equations for Single-LFSR Scrambler

Lane Numbers	Tap Equation
0, 8, 16, 24	D9 xor D13
1, 9, 17, 25	D1 xor D13
2, 10, 18, 26	D13 xor D22
3, 11, 19, 27	D1 xor D22
4, 12, 20, 28	D3 xor D22
5, 13, 21, 29	D1 xor D3
6, 14, 22, 30	D3 xor D9
7, 15, 23, 31	D1 xor D9

### Scrambling Rules

The Gen3 scrambler LFSRs (whether one or more) do not continually advance, but only advance based on what is being sent. The scramblers must be re-initialized periodically and that takes place whenever an EIEOS or FTSOS is seen. The spec gives several rules for scrambling that are listed here for convenience:

# PCI Express Technology

---

- Sync Header bits are not scrambled and do not advance the LFSR.
- The Transmitter LFSR is reset when the last EIEOS Symbol has been sent, and the Receiver LFSR is reset when the last EIEOS Symbol is received.
- TS1 and TS2 Ordered Sets:
  - Symbol 0 bypasses scrambling
  - Symbols 1 to 13 are scrambled
  - Symbols 14 and 15 may or may not be scrambled. The spec states that they will bypass scrambling if necessary to improve DC Balance, but otherwise will be scrambled (see “TS1 and TS2 Ordered Sets” on page 510 for more details on how DC Balance is maintained).
- All Symbols of the Ordered Sets FTS, SDS, EIEOS, EIOS, and SOS bypass scrambling. Despite this, the output data stream will have sufficient transition density to allow clock recovery and the symbols chosen for the Ordered Sets result in a DC balanced output.
- Even when bypassed, Transmitters advance their LFSRs for all Ordered Set Symbols except for those in the SOS.
- Receivers do the same, checking Symbol 0 of an incoming Ordered Set to see whether it is an SOS. If so, the LFSRs are not advanced for any of the Symbols in that Block. Otherwise the LFSRs are advanced for all the Symbols in that Block.
- All Data Block Symbols are scrambled and advance the LFSRs.
- Symbols are scrambled in little-endian order, meaning the least-significant bit is scrambled first and the most-significant bit is scrambled last.
- The seed value for a per-Lane LFSR depends on the Lane number assigned to the Lane when the LTSSM first entered Configuration.Idle (having finished the Polling state). The seed values, modulo 8, are shown in Table 12-3 on page 432 and, once assigned, won’t change as long LinkUp = 1 even if Lane assignments are changed by going back to the Configuration state.
- Unlike 8b/10b, scrambling cannot be disabled while using 128b/130b encoding because it is needed to help with signal integrity. It’s not expected that the Link would operate reliably without it, so it must always be on.
- A Loopback Slave must not scramble or de-scramble the looped-back bit.

---

## Serializer

This shift register works like it does for Gen1/Gen2 data rates except that it is now receiving 8 bits at a time instead of 10 (i.e., the serializer is an 8-bit parallel to serial shift register).

### Mux for Sync Header Bits

Finally, the two Sync Header bits must be injected to distinguish the next Block of characters as a Data Block or an Ordered Set Block. These are the first two bits of each 130-bit Block and the logic for them could be added anywhere in the transmitter that makes sense for the design. In this example the bits are injected at the end of the process for simplicity. Wherever they are included, the flow of bytes from above must be stalled to allow time for them. In this example there will need to be a way to inform the logic above to pause for two bit times. The flow of incoming packets will just be queued in the Tx Buffer during the time the Sync bits are being sent.

---

### Gen3 Physical Layer Receive Logic

As in the earlier generations, the Receiver's logic, shown in Figure 12-18 on page 436, begins with the CDR (Clock and Data Recovery) circuit. This probably includes a PLL that locks onto the frequency of the Transmitter clock based on knowledge of the expected frequency and the edges in the bit stream to generate a recovered clock (Rx Clock). This recovered clock latches the incoming bits into a deserializing buffer and then, once Block Alignment has been established (during the Recovery state of the LTSSM), another version of the recovered clock that is divided by 8.125 (Rx Clock/8.125) latches the 8-bit Symbols into the Elastic Buffer. After that, the de-scrambler recreates the original data from the scrambled characters. The bytes bypass the 8b/10b decoder and are delivered directly to the Byte Un-striping logic. Finally, the Ordered Sets are filtered out, and the remaining byte stream of TLPs and DLLPs is forwarded up to the Data Link Layer.

In the following discussion, each part is described working upward from the bottom. The focus is on describing aspects of the Physical Layer changed for 8.0 GT/s. Sub-block unchanged from Gen1/Gen2 will not be described in this section.

---

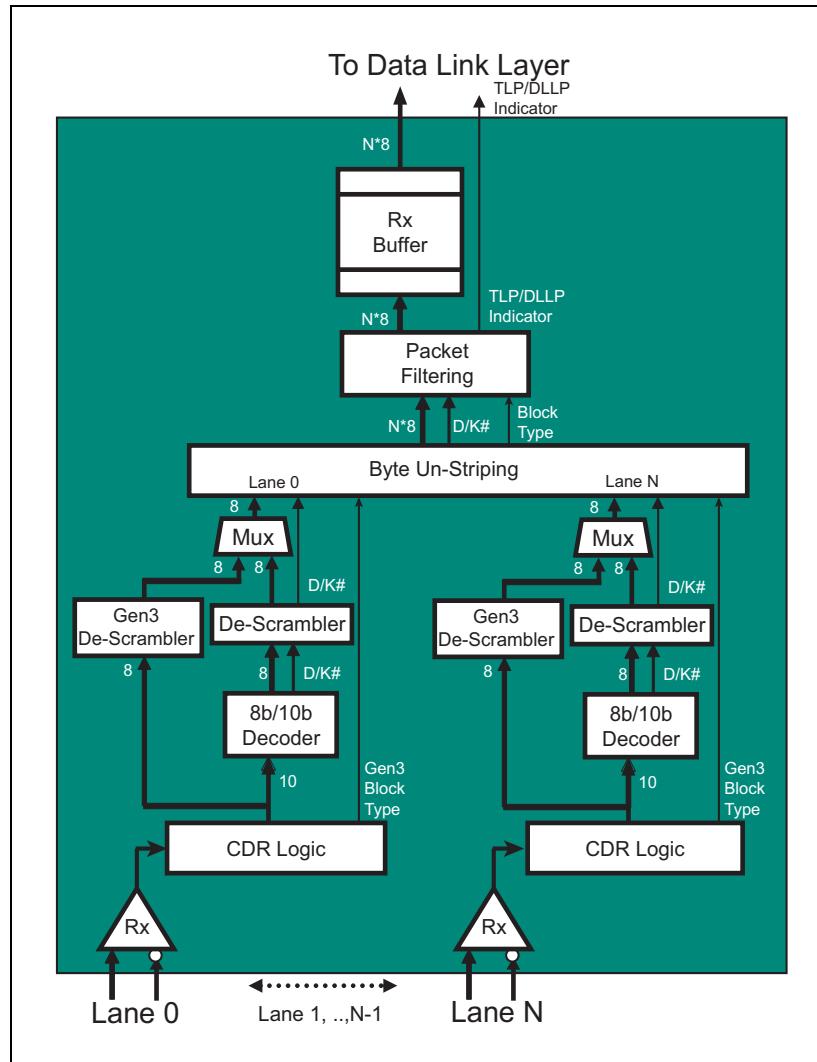
### Differential Receiver

The differential receiver logic is unchanged, but there are electrical changes to improve signal integrity (see "Signal Compensation" on page 468), as well as training changes to establish signal equalization, which are covered in "Link Equalization Overview" on page 577.

# PCI Express Technology

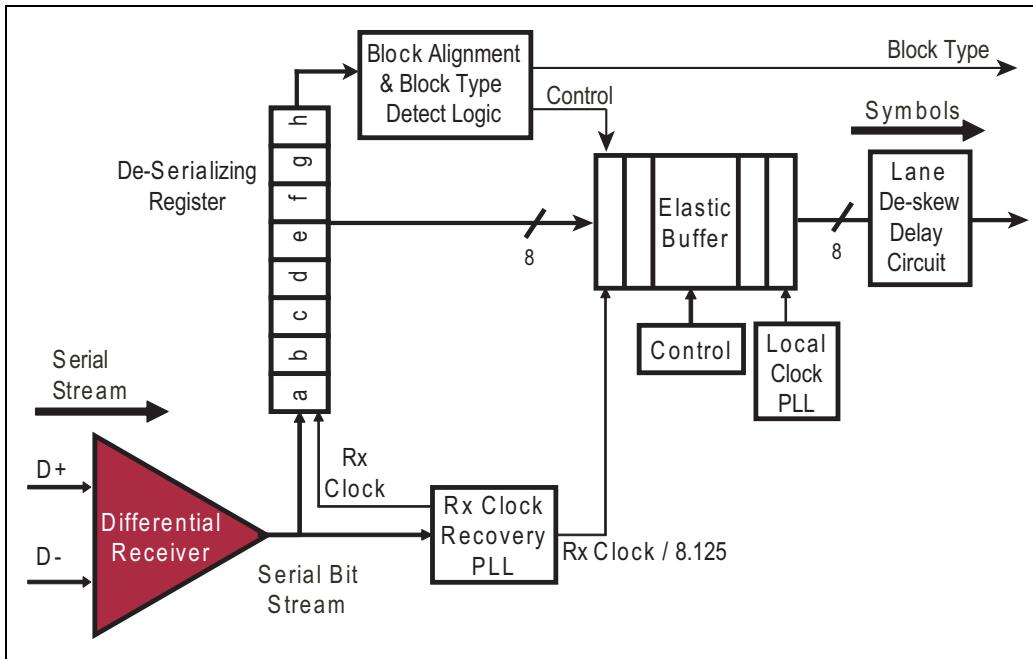
---

Figure 12-18: Gen3 Physical Layer Receiver Details



## Chapter 12: Physical Layer - Logical (Gen3)

Figure 12-19: Gen3 CDR Logic



### CDR (Clock and Data Recovery) Logic

#### Rx Clock Recovery

Although the new scrambling scheme helps with clock recovery, it doesn't guarantee good transition density over short intervals. As a result, the CDR logic must now be able to maintain synchronization for longer periods without as many edges. No specific method for accomplishing this is given in the spec, but a more robust PLL (Phase-Locked Loop) or DLL (Delay-Locked Loop) circuit will likely be needed.

Another aspect of the CDR logic that's different now is that the internal clock used by the Elastic Buffer is not simply the Rx clock divided by 8 as one might expect. The reason, of course, is that the input is not a regular multiple of 8-bit bytes. Instead, it is a 2-bit Sync Header followed by 16 bytes. Those extra two bits must be accounted for somewhere. The spec doesn't require any particular implementation, but one solution would have the clock divided by 8.125, as shown in Figure 12-19 on page 437, to produce 16 clock edges over 130 bit times.

# PCI Express Technology

---

The Block Type Detection logic might then be used to take the extra two bits out of the deserializer that it needs to examine anyway, when a block boundary time is reached, ensuring that only 8-bit bytes are delivered to the Elastic Buffer.

Just to tie up all the loose ends on this discussion, the internal clock for the 8.0 GT/s data rate will actually be  $8.0 \text{ GHz} / 8.125 = 0.985 \text{ GHz}$ . That results in slightly less than the 1.0 GB/s data rate that's usually used to describe the Gen3 bandwidth, but the difference is small enough (1.5% less than 1 GB/s) that it usually isn't mentioned.

## Deserializer

The incoming data is clocked into each Lane's serial-to-parallel converter by the recovered Rx clock, as shown in Figure 12-19 on page 437. The 8-bit Symbols are sent to the Elastic Buffer and clocked into the Elastic Buffer by a version of the Rx Clock that has been divided by 8.125 to properly accommodate 16 bytes in 130 bits.

## Achieving Block Alignment

The EIEOSs sent during training serve to identify boundaries for the 130-bit blocks. As shown in Figure 12-20 on page 438, this Ordered Set can be recognized in a bit stream because it appears as a pattern of alternating bytes of 00h and FFh. When this pattern is seen, the last Symbol of the EIEOS is interpreted as the Block boundary, and testing the next 130 bits will reveal whether the boundary is correct. If not, the logic continues to search for this pattern. This process is described in the spec as occurring in three phases: Unaligned, Aligned, and Locked.

Figure 12-20: EIEOS Symbol Pattern

0	00000000
1	11111111
2	00000000
3	11111111
4	00000000
⋮	
13	11111111
14	00000000
15	11111111

## Chapter 12: Physical Layer - Logical (Gen3)

---

**Unaligned Phase.** Receivers enter this phase after a period of Electrical Idle, such as after changing to 8.0 GT/s or exiting from a low-power Link state. In this phase, the Block Alignment logic watches for the arrival of an EIEOS, since the end of the alternating bytes must correspond to the end of the Block. When an EIEOS is seen, the alignment is adjusted and the logic proceeds to the next phase. Until then, it must also adjust its Block alignment based on the arrival of any SOS.

**Aligned Phase.** In this phase Receivers continues to monitor for EIEOS and make any necessary adjustments to their bit and Block alignment if they see one. However, since they've tentatively identified block boundaries they can also now search for an SDS (Start of Data Stream) Ordered Set to indicate the beginning of a Data Stream. When an SDS is seen, the receiver proceeds to the Locked phase. Until then, it must also adjust its Block alignment based on the arrival of SOSs. If an undefined Sync Header is detected (value of 00b or 11b) the Receiver is allowed to return to the Unaligned phase. The spec notes that this will happen during Link training when EIEOS is followed by a TS Ordered Set.

**Locked Phase.** Once a Receiver reaches this phase, it no longer adjusts its Block alignment. Instead, it now expects to see a Data Block after the SDS and if the alignment has to be readjusted at this point, some misaligned data will probably be lost. If an undefined Sync Header is detected the Receiver is allowed to return to the Unaligned or Aligned phase. Receivers can be directed to transition out of the Locked phase to one of the others as long as Data Stream processing is stopped (see "Data Stream and Data Blocks" on page 413 for the rules regarding Data Streams).

**Special Case: Loopback.** While discussing Block alignment, the spec describes what happens when the Link is in Loopback mode. The Loopback Master must be able to adjust alignment during Loopback, and is allowed to send EIEOS and adjust its Receiver based on a detected EIEOS when they are echoed back during Loopback.Active. The Loopback Slave must be able to adjust alignment during Loopback.Entry but must not adjust alignment during Loopback.Active. The Slave's Receiver is considered to be in the Locked phase when the Slave begins to loop back the bit stream.

### Block Type Detection

Once Block Alignment has been achieved, the Receiver can recognize the start times of the incoming blocks and examine the first two bits to identify which of the two possible types are coming in. Ordered Set Blocks are only interesting to the Physical Layer, so they're not forwarded to the higher layers, but Data

Blocks do get forwarded. When the Sync Header is detected, this information is signaled to other parts of the Physical Layer to determine whether the current block should be removed from the byte stream going to the higher layers. The clock recovery mechanism and Sync Header detection effectively accomplishes the conversion from 130 bits to 128 bits that must take place in the Physical Layer.

Note that since the block information is the same for every Lane, this logic may simply be implemented for only one Lane, such as Lane 0 as shown in Figure 12-18 on page 436. However, if different Link widths and Lane Reversal were supported then more Lanes would need to include this logic to ensure that there would always be one active Lane with this logic available. An example might be that every Lane which is able to operate as Lane 0 would implement it, but only the one that was currently acting as Lane 0 would use it. Note also that, since the spec doesn't give details in this regard, the examples discussed and illustrated here are only educated guesses at a workable implementation.

---

## Receiver Clock Compensation Logic

### Background

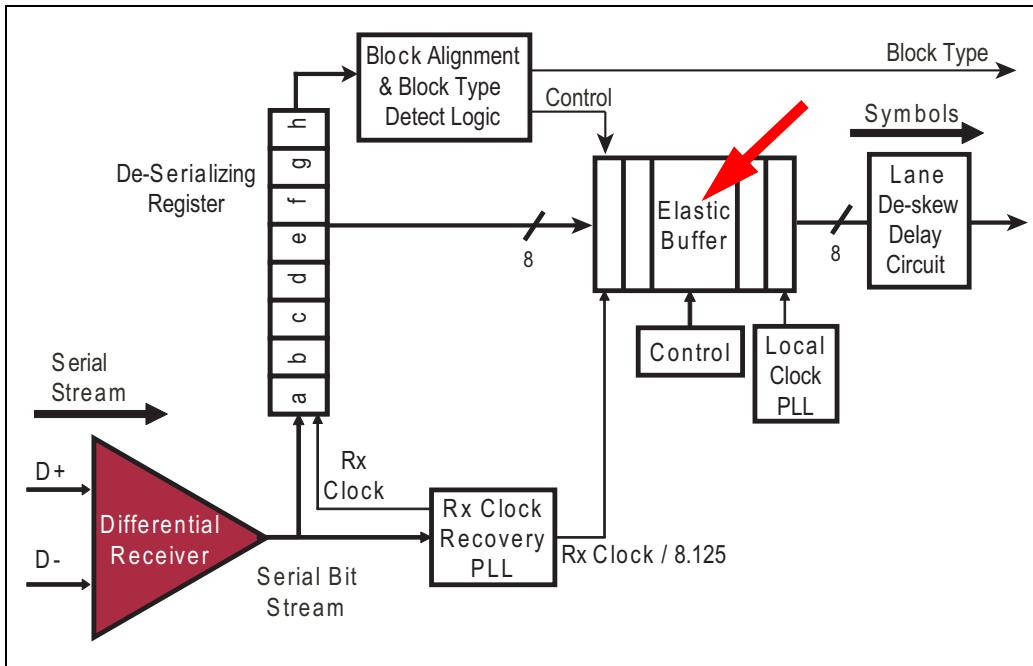
The clock requirements for 8.0 GT/s are the same as they were in the earlier spec versions: the clocks of both Link partners must be within  $\pm 300$  ppm (parts per million) of the center frequency, which results (in the worst case) in gaining or losing one clock after every 1666 clocks.

### Elastic Buffer's Role

The received Symbols are clocked into the elastic buffer, as shown in Figure 12-21 on page 441, using the recovered clock and clocked out using the receiver's local clock. The Elastic Buffer compensates for the frequency difference by adding or removing SKP Symbols as before, but now it does so four Symbols at a time instead of only one at a time. When a SKP Ordered Set arrives, control logic watching the status of the buffer makes an evaluation. If the local clock is running faster, the buffer will be approaching an underflow condition and the logic can compensate by appending four extra SKPs when the SOS arrives to quickly refill the buffer. On the other hand, if the recovered clock is running faster, the buffer will be approaching an overflow condition and the logic will compensate for that by deleting four SKPs to quickly drain the buffer when an SOS is seen.

## Chapter 12: Physical Layer - Logical (Gen3)

Figure 12-21: Gen3 Elastic Buffer Logic



Gen3 Transmitters schedule an SOS once every 370 to 375 blocks but, as before, they can only be sent on block boundaries. If a packet is in progress when SOSs are scheduled, they are accumulated and inserted at the next packet boundary. However, unlike the lower data rates, two consecutive SOSs are not allowed at 8.0 GT/s; they must be separated by a Data Block. Receivers must be able to tolerate SOSs separated by the maximum packet payload size a device supports.

The fact that adjustments are only made in increments of 4 Symbols may affect the depth of the Elastic Buffer, since a difference of 4 would need to be seen before any compensation is applied, and a large packet may be in progress at what would otherwise be the appropriate time. For that reason, care will need to be exercised in determining the optimal size of this buffer, so let's consider an example. The allowed time between SOSs of 375 blocks at 16 Symbols per block equals 6000 Symbol times. Dividing that by the worst-case time to gain or lose a clock of 1666 means that 3.6 clocks could be gained or lost during that period. If the largest possible TLP (4KB) had started just prior to the next SOS being sent, the overall delay for it becomes about  $6000 + 4096 = 10096$  Symbol times for a x1 Link, which translates to a gain or loss of  $10096 / 1666 = 6.06$  clocks. Conse-

quently, if TLPs of 4KB in size are supported, the buffer might be designed to handle 7 Symbols too many or too few before an SOS is guaranteed to arrive. It may happen that two SOSs are scheduled before the first one is sent. At the lower data rates, the queued SOSs are sent back-to-back, but for 8.0 GT/s they are not and must be separated by a Data Block. Whenever an SOS does arrive at the Receiver, it can add or remove 4 SKP Symbols to quickly fill or drain the buffer and avoid a problem.

---

## Lane-to-Lane Skew

### Flight Time Variance Between Lanes

For multi-Lane Links, the difference in arrival times between lanes is automatically corrected at the Receiver by delaying the early arrivals until they all match up. The spec allows this to be accomplished by any means a designer prefers, but using a digital delay after the elastic buffer has one advantage in that the arrival time differences are now digitized to the local Symbol clock of the receiver. If the input to one lane makes it on a clock edge and another one doesn't, the difference between them will be measured in clock periods, so the early arrival can simply be delayed by the appropriate number of clocks to get it to line up with the late-comers (see Figure 12-22 on page 444). The fact that the maximum allowable skew at the receiver is a multiple of the clock periods makes this easy and infers that the spec writers may have had this implementation in mind. As defined in the spec, the receiver must be capable of de-skewing up to 20ns for Gen1 (5 Symbol-time clocks at 4ns per Symbol) and 8ns for Gen2 (4 Symbol-time clocks at 2ns per Symbol), and 6ns for Gen3 (6 Symbol-time clocks at 1ns per Symbol).

### De-skew Opportunities

The same Symbol must be seen on all lanes at the same time to perform de-skewing, and any Ordered Set will do. However, de-skewing is only performed in the L0s, Recovery, and Configuration LTSSM states. In particular, it must be completed as a condition for:

- Leaving Configuration.Complete
- Beginning to process a Data Stream after leaving Configuration.Idle or Recovery.Idle
- Leaving Recovery.RcvrCfg
- Leaving Rx\_L0s.FTS

## Chapter 12: Physical Layer - Logical (Gen3)

---

If skew values change while in L0 (based on temperature or voltage changes, for example), a Receiver error may occur and cause replayed TLPs. If the problem becomes persistent, the Link would eventually transition to the Recovery state and de-skewing would take place there. The spec notes that, while devices are not allowed to de-skew their Lanes while in L0, the SOSs that must be sent periodically in this state contain an LFSR value that is intended to aid external tools in doing this. These tools, unconstrained by the rules for Data Streams, can search for the SOSs and use the patterns to achieve Bit Lock, Block Alignment and Lane-to-Lane de-skew in the midst of a Data Stream.

The spec notes that when leaving L0s the Transmitter will send an EIEOS, then the correct number of FTSs with another EIEOS inserted after every 32 FTSs, then one last EIEOS to assist with Block Alignment and, finally, an SDS Ordered Set for the purpose of de-skewing in addition to starting the Data Stream.

### Receiver Lane-to-Lane De-skew Capability

Understandably, the transmitter is only allowed to introduce a minimal amount of skew so as to leave the rest of the skew budget to cover routing differences and other variations. The amount of allowed skew that can be corrected at the Receiver is shown in Table 12-5 on page 443, where it can be seen that this skew corresponds easily to a number of Symbol times for Gen3 just as it did for the earlier data rates. That allows the same option of using delay registers to accomplish de-skew after the elastic buffer as was described for Gen1/Gen2 Physical Layer implementations earlier.

Table 12-5: Signal Skew Parameters

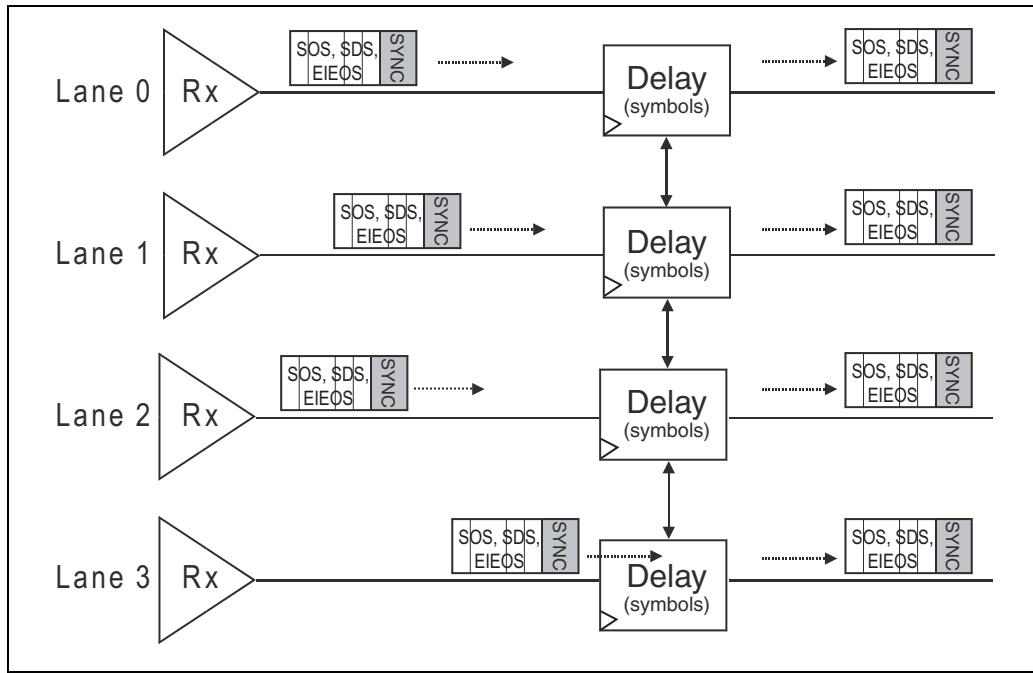
	Gen1	Gen2	Gen3
Tx max skew	1.3 ns	1.3 ns	1.1 ns
Rx max skew	20 ns	8 ns	6 ns
Symbol time period	4ns	2ns	1ns
Rx skew expressed in Symbol Times	5	4	6

When using 8b/10b encoding, an unambiguous de-skew mechanism is to watch for the COM control character, which must appear on all Lanes simultaneously. That option is not available for 128b/130b, but Ordered Sets still arrive at the same time on all the Lanes, such as the SOS, SDS, and EIEOS. As a result, the process can be very much the same even though the pattern to search for when de-skewing the Lanes is different.

# PCI Express Technology

---

Figure 12-22: Receiver Link De-Skew Logic



---

## Descrambler

### General

Receivers follow exactly the same rules for generating the scrambling polynomial that the Transmitter does and simply XOR the same value to the input data a second time to recover the original information. Like on the transmit side, they are allowed to implement a separate LFSR for each Lane or just one.

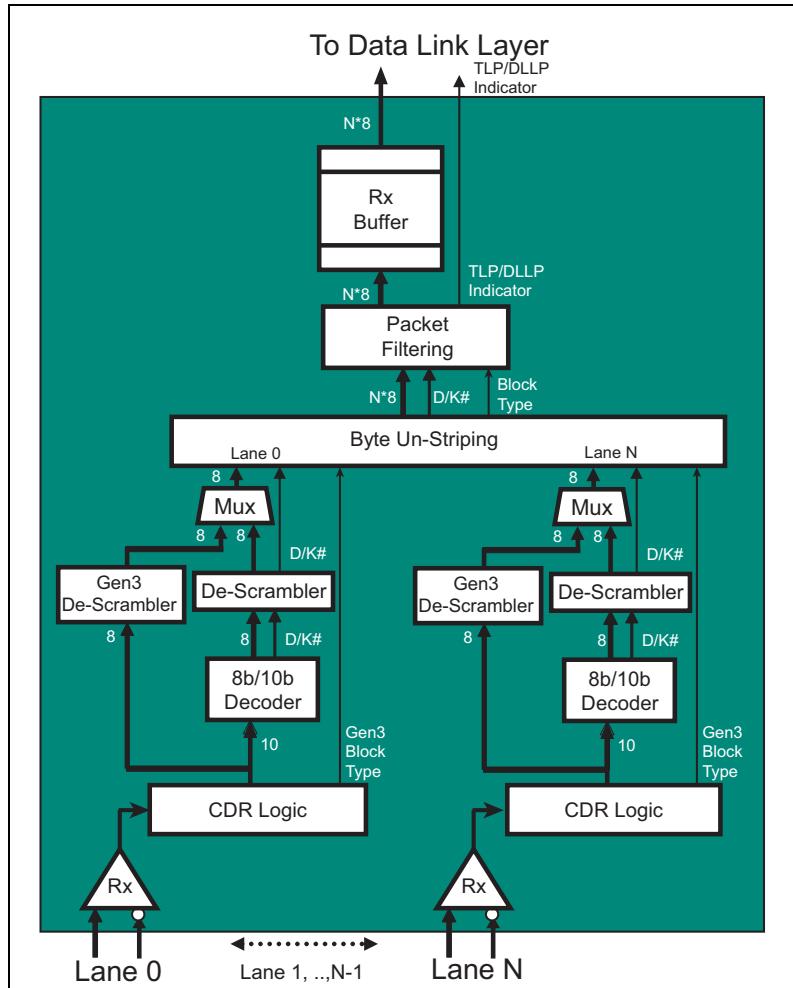
### Disabling Descrambling

Unlike at Gen1/Gen2 data rates, in Gen3 mode, descrambling cannot be disabled because of its role in facilitating clock recovery and signal integrity. At the lower rates, the “disable scrambling” bit in the control byte of TS1s and TS2s would be used to inform a Link neighbor that scrambling was being turned off. That bit is reserved for rates of 8.0 GT/s and higher.

## Byte Un-Striping

This logic is basically unchanged from Gen1 or Gen2 implementation. At some point, the byte streams for Gen3 and for the lower data rates will have to muxed together, and the example in Figure 12-23 on page 445 shows that happening just before the un-striping logic.

Figure 12-23: Physical Layer Receive Logic Details



---

## Packet Filtering

The serial byte stream supplied by the byte un-striping logic contains TLPs, DLLPs, Logical Idles (IDLs), and Ordered Sets. The Logical Idle bytes and Ordered Sets are eliminated here and are not forwarded to the Data Link layer. What remains are the TLPs and DLLPs, which get forwarded along with an indicator of their packet type.

---

## Receive Buffer (Rx Buffer)

The Rx Buffer holds received TLPs and DLLPs until the Data Link Layer is able to accept them. The interface to the Data Link Layer is not described in the spec, and so a designer is free to choose details like the width of this bus. The wider the path, the lower the clock frequency will be, but more signals and logic will be needed to support it.

---

## Notes Regarding Loopback with 128b/130b

The spec makes a special point to describe the operation of Loopback Mode at the higher rate. The basic rules can be summarized as follows:

- Loopback masters must send actual Ordered Sets or Data Blocks, but they aren't required to follow the normal protocol rules when changing from Data Blocks to Ordered Sets or vice versa. In other words, the SDS Ordered Set and EDS token are not required. Slaves must not expect or check for the presence of them.
- Masters must send SOS as usual, and must allow for the number of SKP Symbols in the loopback stream to be different because the receiver will be performing clock compensation.
- Loopback slaves are allowed to modify the SOS by adding or removing 4 SKP Symbols at a time as they normally would for clock compensation, but the resulting SOS must still follow the proper format rules.
- Everything should be looped back exactly as it was sent except for SOS which can change as just described, and both EIEOS and EIOS which have defined purposes in loopback and should be avoided.
- If a slave is unable to acquire Block alignment, it won't be able to loop back all bits as received and is allowed to add or remove Symbols as needed to continue operation.

---

# 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 ( $Z_{RX-HIGH-IMP-DC-POS}$  and  $Z_{RX-HIGH-IMP-DC-NEG}$ ) 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.
- **$V_{TX-ACCM}$  and  $V_{RX-ACCM}$ :** For 2.5 and 5.0 GT/s these are relaxed to 150 mVPP for the Transmitter and 300 mVPP for the Receiver.

# **Chapter 13: Physical Layer - Electrical**

---

---

## **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*.

---

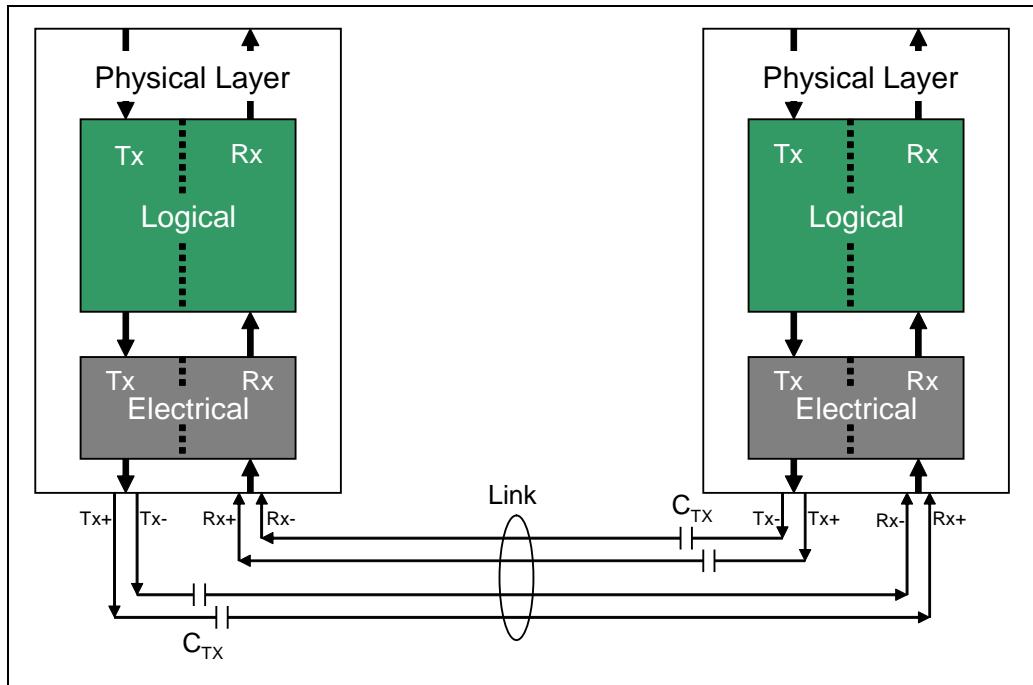
## **Physical Layer Electrical Overview**

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.

# PCI Express Technology

---

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-

# Chapter 13: Physical Layer - Electrical

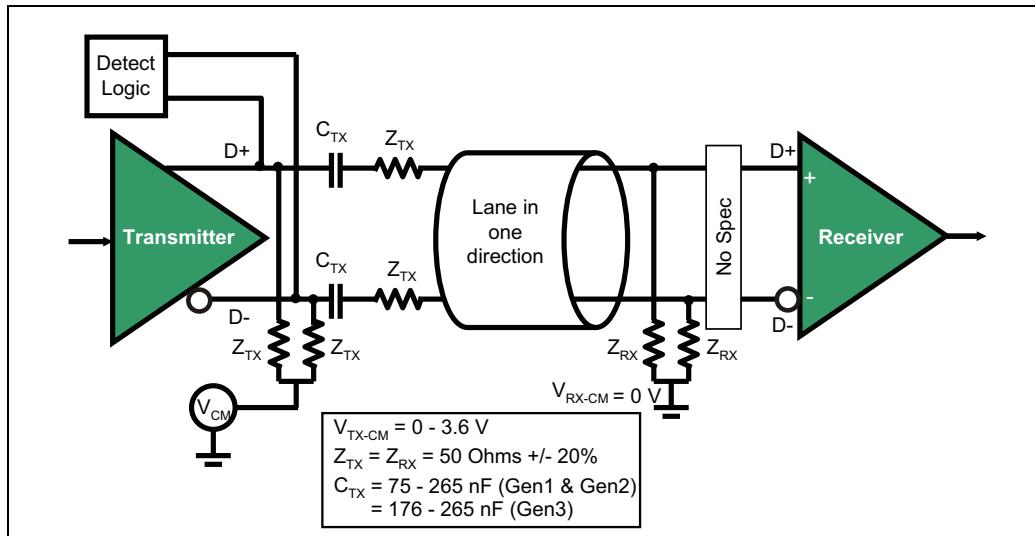
couple the DC part of the signal between the Link partners and means they don't have to share a common power supply or ground return path, as when the devices are connected over a cable. Figure 13-1 on page 450 illustrates the placement of this capacitor ( $C_{TX}$ ) on the Link.

## High Speed Signaling

The high-speed signaling environment of PCIe is characterized by the drawing in Figure 13-2 on page 451. This low-voltage differential signaling environment is a common method used in many serial transports and one reason is for the noise rejection it provides. Electrical noise that affects one signal will also affect the other because they are on adjacent pins and their traces are very close to each other. Since both signals are influenced, as shown in Figure 13-3 on page 452, the difference between them doesn't change much and is therefore not seen at the receiver.

A design goal for the 3.0 spec revision was that the 8.0 GT/s rate should still work with existing standard FR4 circuit boards and connectors, and that was achieved by changing the encoding scheme from the old 8b/10b to the new 128b/130b model to keep the frequency low. This goal will probably change with the next speed step (Gen4).

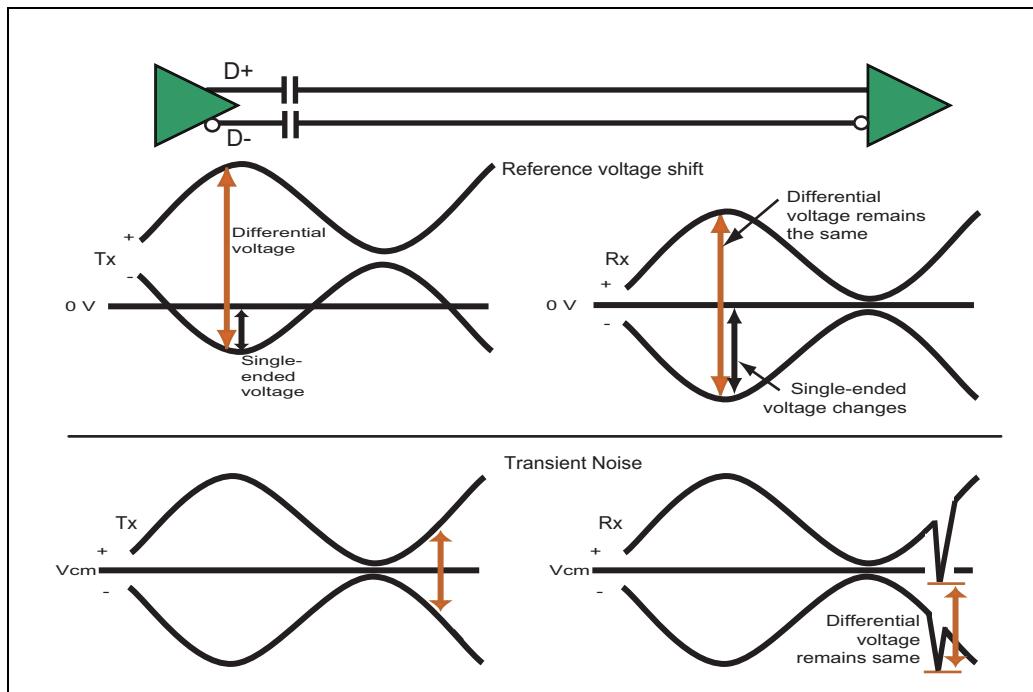
Figure 13-2: Differential Transmitter/Receiver



# PCI Express Technology

---

Figure 13-3: Differential Common-Mode Noise Rejection



---

## Clock Requirements

---

### General

For all data rates, both Transmitter and Receiver clocks must be accurate to within  $\pm 300$  ppm (parts per million) of the center frequency. In the worst case, the Transmitter and Receiver could both be off by 300 ppm in opposite directions, resulting in a maximum difference of 600 ppm. That worst-case model translates to a gain or loss of 1 clock every 1666 clocks and that's the difference that a Receiver's clock compensation logic must take into account.

Devices are allowed to derive their clocks from an external source, and the 100 MHz Refclk is still optionally available for this purpose in the 3.0 spec. Using the Refclk permits both Link partners to readily maintain the 600 ppm accuracy even when Spread Spectrum Clocking is applied.

### SSC (Spread Spectrum Clocking)

SSC is an optional technique used to modulate the clock frequency slowly over a prescribed range to spread the signal's EMI (Electro-Magnetic Interference) across a range of frequencies rather than allowing it all to be concentrated at the center frequency. Spreading the radiated energy helps a device or system to pass government emissions standards by staying under a threshold value, as illustrated in Figure 13-4 on page 454. Note that the frequency of interest for the signal is only half the clock rate because two rising clock edges are needed to create one cycle on the data, as illustrated in Figure 13-5 on page 454. For example, a 2.5 GT/s rate uses a bit clock of 2.5 GHz, resulting in a frequency of interest on the traces of 1.25 GHz.

The use of SCC is not required by the spec but, if will be supported, the following rules apply:

- The clock can be modulated by +0% to -0.5% from nominal (5000 ppm), referred to as "down spreading." A frequency modulation envelope is not specified, but a sawtooth-wave pattern like the one shown in Figure 13-6 on page 455 yields good results. Note that there is a trade-off with down spreading, because the average clock frequency will now be 0.25% lower than it would have been without SSC, resulting in a slight performance reduction.
- The modulation rate must be between 30KHz and 33KHz.
- The +/- 300 ppm requirement for clock frequency accuracy still holds and therefore so does the maximum 600 ppm variation between Link partners. The spec states that most implementations will require both Link partners to use the same clock source, although it's not required. One way to do that would be for them to both use a modulated version of the Refclk to derive their own clocks (see "Common Refclk" on page 456).

# PCI Express Technology

---

Figure 13-4: SSC Motivation

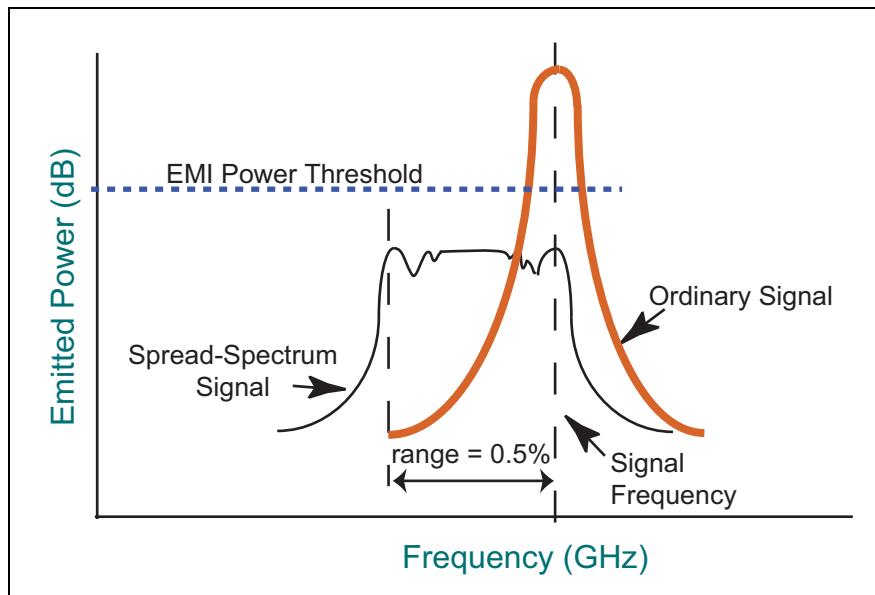


Figure 13-5: Signal Rate Less Than Half the Clock Rate

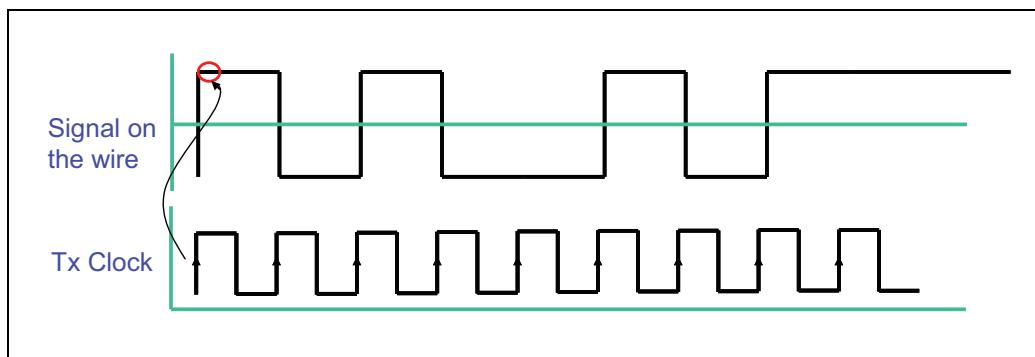
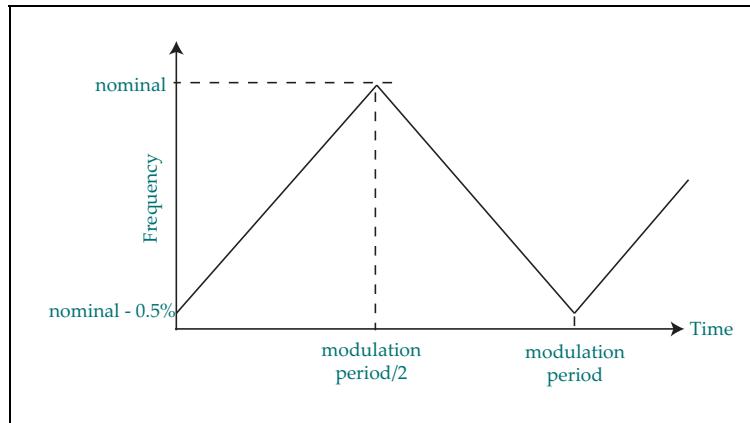


Figure 13-6: SSC Modulation Example



---

## Refclk Overview

Receivers must generate their own clocks to operate their internal logic, but there are some options for generating the recovered clock for the incoming bit stream. The details for them have developed with each succeeding version of the spec and are based on the data rate.

### 2.5 GT/s

In the early spec versions using the 2.5 GT/s rate, information regarding the optional Refclk was not included in the base spec but instead in the separate CEM (Card Electro-Mechanical) spec for PCIe. A number of parameters were specified there and several general terms have been carried forward to the newer versions of the spec. The Refclk was described as a 100 MHz differential clock driving a  $100\ \Omega$  differential load (+/- 10%) with a trace length limited to 4 inches. SSC is allowed, as described in "SSC (Spread Spectrum Clocking)" on page 453.

### 5.0 GT/s

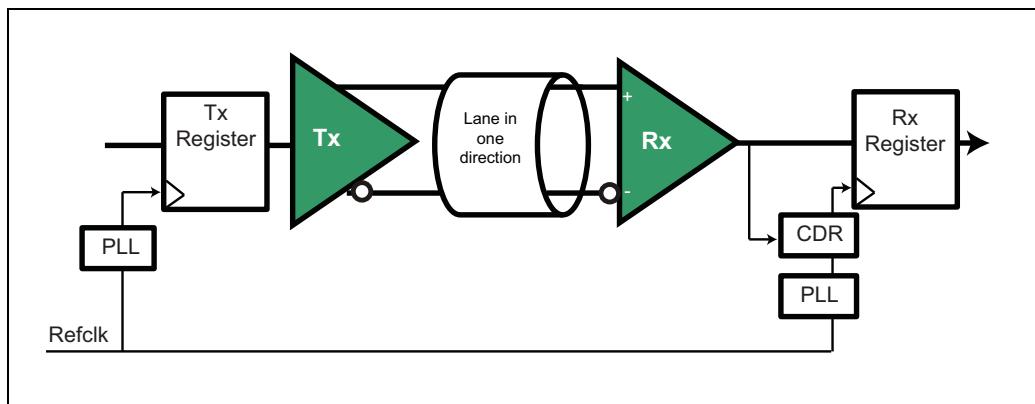
When the 5.0 GT/s rate was developed, the spec writers chose to include the Refclk information in the electrical section of the base spec and listed three options for the clock architecture:

# PCI Express Technology

**Common Refclk.** The first architecture described is one in which both Link partners make use of the same Refclk, as shown in Figure 13-7 on page 456. There are three straightforward advantages for this implementation:

- First, the jitter associated with the reference clock is the same for both Tx and Rx and is thus tracked and accounted for intrinsically.
- Second, the use of SSC will be simplest with this model because maintaining the 600 ppm separation between the Tx and Rx clocks is easy if both follow the same modulated reference.
- Third, the Refclk remains available during low-power Link states L0s and L1 and that allows the Receiver's CDR to maintain a semblance of the recovered clock even in the absence of a bit stream to supply the edges in the data. That, in turn, keeps the local PLLs from drifting as much as they otherwise would, resulting in a reduced recovery time back to L0 compared to the other clocking options.

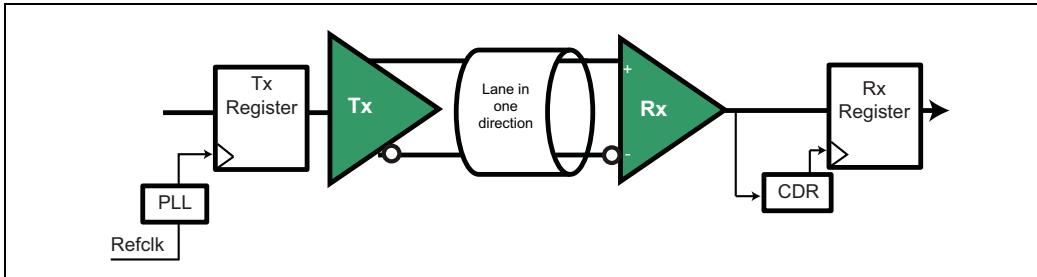
Figure 13-7: Shared Refclk Architecture



**Data Clocked Rx Architecture.** In this clock architecture, the Receiver doesn't use a reference clock at all, but simply recovers the Transmitter clock from the data stream, as shown in Figure 13-9 on page 457. This implementation is clearly the simplest of the three and would therefore ordinarily be preferred. The spec doesn't prohibit the use of SSC in this model, but doing so would bring up two issues. First, the Receiver CDR must remain locked onto the input frequency as it modulates through a much wider range (5600 ppm instead of the usual 600 ppm), and that could require more complex logic. And second, the maximum clock frequency separation of 600ppm must still be maintained and it's less clear how that would be done without a common reference.

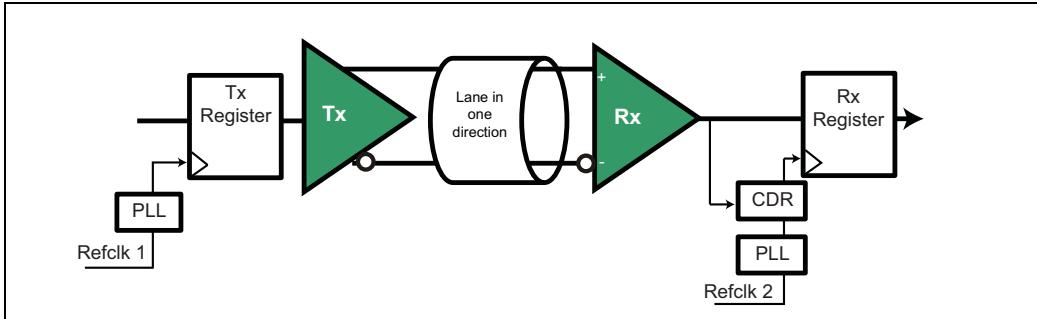
## Chapter 13: Physical Layer - Electrical

Figure 13-8: Data Clocked Rx Architecture



**Separate Refclks.** Finally, it's also possible for the Link partners to use different reference clocks, as shown in Figure 13-9 on page 457. However, this implementation makes substantially tighter demands on the Refclks because the jitter seen at the Receiver will be the RSS (Root Sum of Squares) combination of them both, making the timing budget difficult. It also becomes enormously more difficult to manage SSC in this model and that's why the spec states that SSC must be turned off in this case. Overall, the spec gives the impression that this is the least desirable alternative, and states that it doesn't explicitly define the requirements for this architecture.

Figure 13-9: Separate Refclk Architecture



### 8.0 GT/s

The same three clock architectures are described in the spec for this data rate, too. One difference is that two types of CDR are defined now: a 1<sup>st</sup> order CDR for the shared Refclk architecture, and a 2<sup>nd</sup> order CDR for the data clocked architecture. This just reflects the fact that, as it was for the lower data rates, the CDR for the data-clocked architecture will need to be more sophisticated to be able to stay locked when the reference varies over a wide range for SSC.

## Transmitter (Tx) Specs

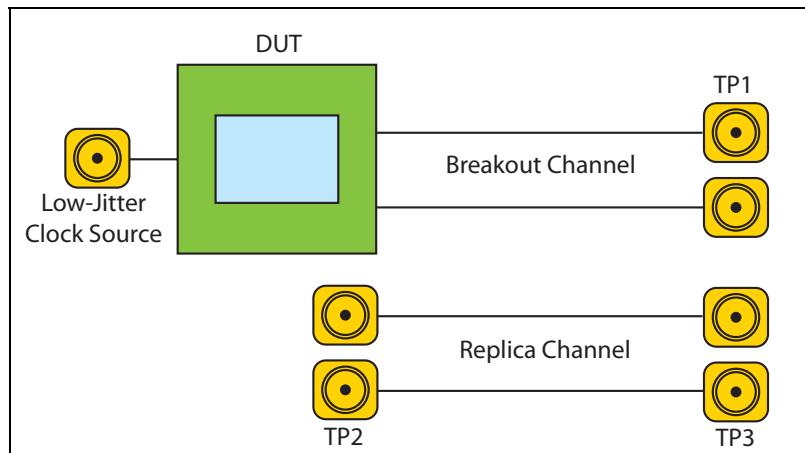
---

### Measuring Tx Signals

The spec notes that the methods for measuring the Tx output are limited at the higher frequencies. At 2.5 GT/s it's possible to put a test probe very near the pins of the DUT (Device Under Test), but for the higher rates it's necessary to use a "breakout channel" with SMA (SubMiniature version A) microwave-type coaxial connectors, as illustrated at TP1 (Test Point 1), TP2, and TP3 in Figure 13-10 on page 458. Note that it's necessary to have a low-jitter clock source to the device under test, so that jitter seen at the output is only introduced by the device itself. The spec also mentions that it's important during testing for the device to have as many of its Lanes and other outputs in use at the same time as possible, so as to best simulate a real system.

Since the breakout channel introduces some effects to the signal, for 8.0 GT/s it's necessary to be able to measure those effects and remove (de-embed) them from the signal being tested. One way to accomplish this is for the test board to supply another signal path that is very similar to the one used for the device pins. Characterizing this "replica channel" with a known signal gives the needed information about the channel, allowing its effects to be de-embedded from the DUT signals so the signal at the component pins can be recovered.

Figure 13-10: Test Circuit Measurement Channels



## Tx Impedance Requirements

For best accuracy, the characteristic differential impedance of the Breakout Channel should be  $100\ \Omega$  differential within 10%, with a single-ended impedance of  $50\ \Omega$ . To match this environment, Transmitters have a differential low-impedance value during signaling between  $80$  and  $120\ \Omega$  at  $2.5\ \text{GT/s}$ , and no more than  $120\ \Omega$  at  $5.0$  and  $8.0\ \text{GT/s}$ . For receivers, the single-ended impedance is  $40$  -  $60\ \Omega$  at  $2.5$  or  $5.0\ \text{GT/s}$ , but for  $8.0\ \text{GT/s}$  no specific value is given. Instead, it's simply noted that the single-ended receiver impedance must be  $50\ \Omega$  within 20% by the time the Detect LTSSM state is entered so that the detect circuit will sense the Receiver correctly.

Transmitters must also meet the return loss parameters  $\text{RL}_{\text{TX-DIFF}}$  and  $\text{RL}_{\text{TX-CM}}$  anytime differential signals are sent. As a very brief introduction to this terminology, "return loss" is a measure of energy transmitted through or reflected back from a transmission path. Return loss is one of several "Scattering" parameters (S-parameters) that are used to analyze high-frequency signal environments. When frequencies are low, a lumped-element description is sufficient, but when they become high enough that the wavelength approaches the size of the circuit, a distributed model is needed and that's what S-parameters are used to represent. The spec describes a number of these to characterize a transmission path but the details of this high-frequency analysis are really beyond the scope of this book.

When a signal is not being driven, as would be the case in the low-power Link states, the Transmitter may go into a high-impedance condition to reduce the power drain. For that case, it only has to meet the  $I_{\text{TX-SHORT}}$  value and the differential impedance is not defined.

---

## ESD and Short Circuit Requirements

All signals and power pins must withstand a  $2000\text{V}$  ESD (Electro-Static Discharge) using the Human Body Model and  $500\text{V}$  using the Charged Device Model. For more details on these models or ESD, see the JEDEC JESE22-A114-A spec.

The ESD requirement not only protects against electro-static damage, but facilitates support of surprise hot insertion and removal events (adding or removing an add-in card while the power is on). That goal also motivates the requirement that Transmitters and Receivers be able to withstand sustained short-circuit currents of  $I_{\text{TX-SHORT}}$  (see Table 13-5 on page 498).

## Receiver Detection

### General

The Detect block in the Transmitter shown in Figure 13-11 on page 461 is used to check whether a Receiver is present at the other end of the Link after coming out of reset. This step is a little unusual in the serial transport world because it's easy enough to send packets to the Link partner and test its presence by whether or not it responds. The motivation for this approach in PCIe, however, is to provide an automatic hardware assist in a test environment. If the proper load is detected, but the Link partner refuses to send TS1s and participate in Link Training, the component will assume that it must be in a test environment and will begin sending the Compliance Pattern to facilitate testing. Since a Link will always start operation at 2.5 GT/s after a reset or power-up event, Detect is only used for the 2.5 GT/s rate. That's why the Receiver's single-ended DC impedance is specified for that rate ( $Z_{RX-DC} = 40$  to  $60 \Omega$ ), and why the Detect logic must be included in every design regardless of its intended operating speed.

Detection is accomplished by setting the Transmitter's DC common mode voltage to one value and then changing it to another. Knowing the expected charge time when a Receiver is present, the logic compares the measured time against that. If a Receiver is attached, the charge time (RC time constant) is relatively long due to the Receiver's termination. Otherwise, the charge time is much shorter

### Detecting Receiver Presence

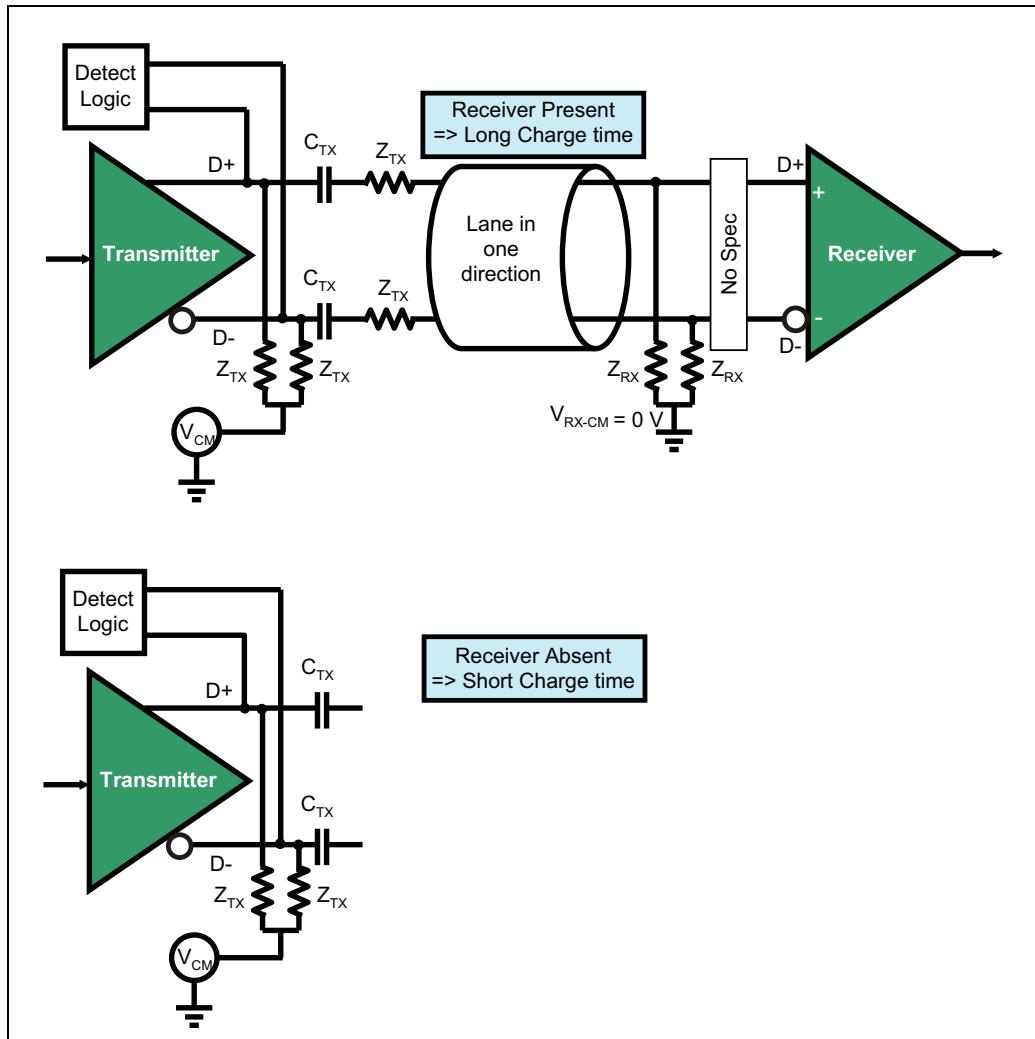
1. After reset or power-up, Transmitters drive a stable voltage on the D+ and D- terminal.
2. Transmitters then change the common mode voltage in a positive direction by no more than the  $V_{TX-RCV-DETECT}$  amount of 600mV specified for all three data rates.
3. Detect logic measures the charge time:
  - Receiver is absent if the charge time is short.
  - Receiver is present if the charge time is long (dominated by the series capacitor and Receiver termination).

The spec mentions a possible problem here: the proper load may appear on one of the differential signals but not the other, and if detection doesn't check both it could misinterpret the situation. The simple way to avoid that would be to perform the Detect operation on both D+ and D-. The 3.0 spec does not require this,

## Chapter 13: Physical Layer - Electrical

but mentions that future spec revisions may. Therefore, it would be wise to include this functionality in new designs.

Figure 13-11: Receiver Detection Mechanism



## Transmitter Voltages

Differential signaling (as opposed to the single-ended signaling employed in PCI and PCI-X) is ideal for high frequency signaling. Some advantages of differential signaling are:

- Receivers look at the difference between the signals, so the voltage swing for each one individually can be smaller, allowing higher frequencies without exceeding the power budget.
- EMI is reduced because of the noise cancellation that results from having the two signals by side by side, using opposite-polarity voltages.
- Noise immunity is very good, because noise that affects one signal will also affect the other in the same way, with the result that the Receiver doesn't notice the change (refer to Figure 13-3 on page 452).

### DC Common Mode Voltage

After the Detect state of Link training, the Transmitter DC common mode voltage  $V_{TX-DC-CM}$  (see Table 13-3 on page 489) must remain at the same voltage. The common mode voltage is turned off only in the L2 or L3 low-power Link states, in which main power to the device is removed. A designer can choose any common mode voltage in the range from 0 to 3.6V.

### Full-Swing Differential Voltage

The Transmitter output consists of two signals, D+ and D-, that are identical but use opposite polarities. A logical one is indicated when the D+ signal is high and the D- signal low, while a logical zero is represented by driving the D+ signal low and the D- signal high, as shown in Figure 13-13 on page 464.

The differential peak-to-peak voltage driven by the Transmitter  $V_{TX-DIFF_{P-P}}$  (see Table 13-3 on page 489) is between 800 mV and 1200 mV (1300 mV for 8.0 GT/s).

- Logical 1 is signaled with a positive differential voltage.
- Logical 0 is signaled with a negative differential voltage.

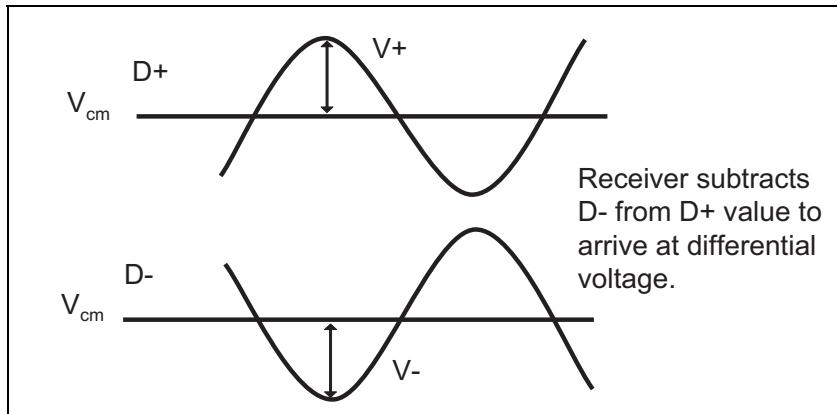
During Electrical Idle the Transmitter holds the differential peak voltage  $V_{TX-IDLE-DIFF_P}$  (see Table 13-3 on page 489) very near zero (0-20 mV). During this time the Transmitter may be in either a low- or high-impedance state.

The Receiver senses a logical one or zero, as well as Electrical Idle, by evaluating the voltage on the Link. The signal loss expected at high frequency means the

# Chapter 13: Physical Layer - Electrical

Receiver must be able to sense an attenuated version of the signal, defined as  $V_{RX-DIFFp-p}$  (see Table 13-5 on page 498).

Figure 13-12: Differential Signaling



## Differential Notation

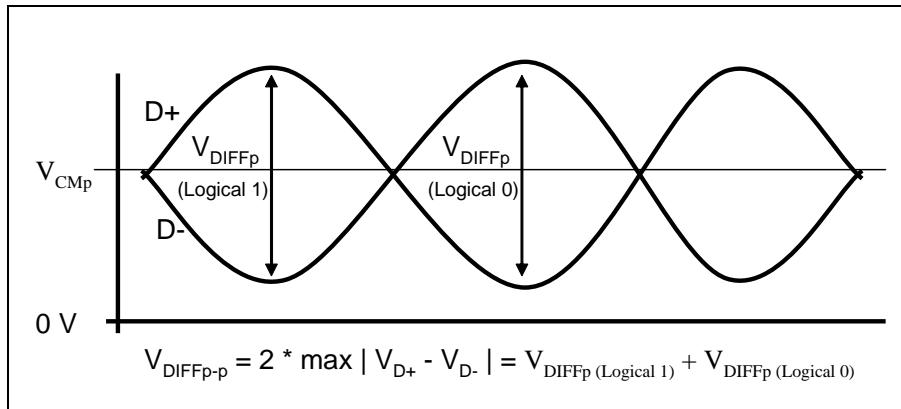
A differential signal voltage is defined by taking the difference in the voltage on the two conductors,  $D+$  and  $D-$ . The voltage with respect to ground on each conductor is  $V_{D+}$  and  $V_{D-}$ . The differential voltage is given by  $V_{DIFF} = V_{D+} - V_{D-}$ . The Common Mode voltage,  $V_{CM}$ , is defined as the voltage around which the signal is switching, which is the mean value given by  $V_{CM} = (V_{D+} + V_{D-}) / 2$ .

The spec uses two terms when discussing differential voltages and confusion sometimes arises as a result. As illustrated in Figure 13-13 on page 464, the Peak value is the maximum voltage difference between the signals, while the Peak-to-Peak voltage is that value plus the maximum in the opposite direction. For a symmetric signal, the Peak-to-Peak value is simply twice the Peak value.

1. Differential Peak Voltage  $\Rightarrow V_{DIFFp} = (\max |V_{D+} - V_{D-}|)$
2. Differential Peak-to-Peak Voltage  $\Rightarrow V_{DIFFp-p} = 2 * (\max |V_{D+} - V_{D-}|)$

As an example, assume  $V_{CM} = 0$  V, then if the  $D+$  value is 300mV and the  $D-$  value is -300mV, then  $V_{DIFFp}$  would be  $300 - (-300) = 600$  mV for a logical one. Similarly, it would be  $(-300) - (+300) = -600$  mV for a logical zero. The  $V_{DIFFp-p}$  for this symmetric case would be 1200 mV. The allowed  $V_{DIFFp-p}$  range for 2.5 GT/s and 5.0 GT/s is 800 to 1200 mV, while for 8.0 GT/s it is 800 to 1300 mV before equalization is applied.

Figure 13-13: Differential Peak-to-Peak ( $V_{\text{DIFFP-P}}$ ) and Peak ( $V_{\text{DIFFP}}$ ) Voltages



## Reduced-Swing Differential Voltage

The full-swing voltage is needed for channels that are long or otherwise lossy, and Transmitters are required to support it. But when the signal environment is short and low loss, a high voltage is unnecessary and a power savings can be realized by reducing it. With this in mind, the spec for 2.5 GT/s and 5.0 GT/s defines another, reduced-swing voltage for power-sensitive systems where a short channel is being used. In this mode the voltage is reduced to about half of its full-swing range. Support for this operation is optional, and the means for selecting it is not defined and will be implementation specific.

The same is true for 8.0 GT/s signaling, except that in this case it's achieved by using a limited range of coefficients. For example, the maximum boost for the reduced-swing case is limited to 3.5 dB. As with the lower data rates, support for this voltage model is optional, but now the means of achieving it is straightforward: just set the Tx coefficient values to make it happen.

It should be noted that the Receiver voltage levels are independent of the transmitter, which is intuitively what we'd expect: the received signal always needs to meet the normal requirements and so the Transmitter and channel must be designed to guarantee that it will.

## Equalized Voltage

In the interest of maintaining a good flow in this section, this large topic is covered separately in the section called "Signal Compensation" on page 468.

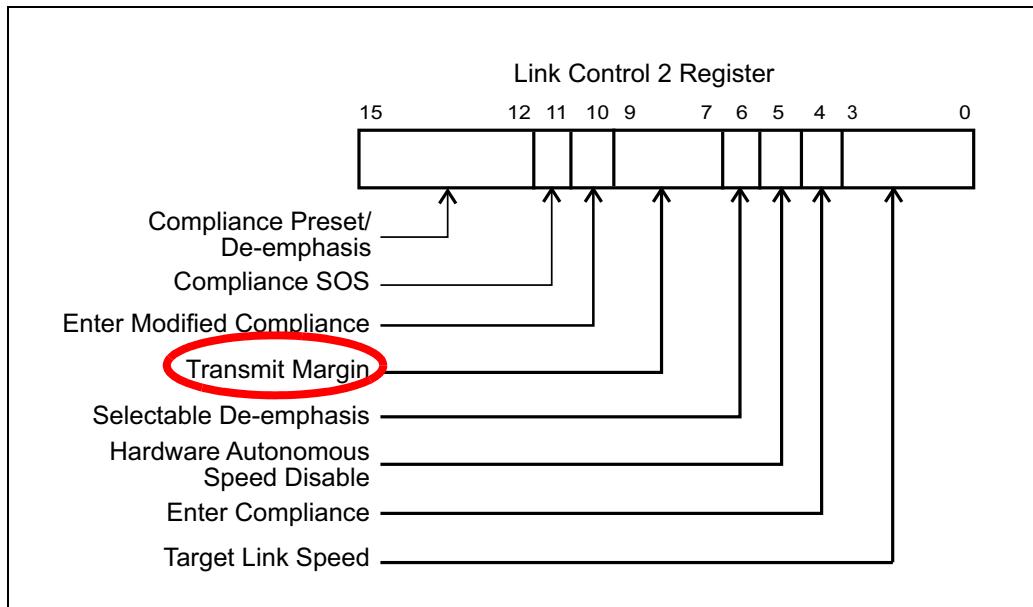
# Chapter 13: Physical Layer - Electrical

## Voltage Margining

The concept of margining is that Transmitter characteristics like output voltage can be adjusted across a wide range of values during testing to determine how well it can handle a signaling environment. The 2.5 GT/s rate didn't include this capability, but voltage margining was added with the 5.0 GT/s rate and must be implemented by Transmitters that use that rate or higher. Other parameters, like de-emphasis or jitter can optionally be margined as well. The granularity for the margining adjustments must be controllable on a Link basis and may be controllable on a Lane basis. This control is accomplished by means of the Link Control 2 register in the PCIe Capability register block. The transmit margin field, shown in Figure 13-14 on page 465, contains 3 bits and can thus represent 8 levels. Their values are not defined, and not all of them need to be implemented. The default value is all zeros, which represents the normal operating range.

It's important to note that this field is only intended for debug and compliance testing purposes during which software is only allowed to modify it during those times. At all other times, the value is required to be set to the default of all zeros.

Figure 13-14: Transmit Margin Field in Link Control 2 Register



For 8.0 GT/s, transmitters are required to implement voltage margining and use the same field in the Link Control 2 register, but equalization adds some constraints to the options because it can't require finer coefficient or preset resolution than the 1/24 resolution defined for normal operation.

During Tx margining the equalization tolerance for 2.5 GT/s and 5.0 GT/s is relaxed from +/- 0.5 dB to +/- 1.0 dB. For the 8.0 GT/s rate, the tolerance is defined by the coefficient granularity and the normal equalizer tolerances specified for the transmitter.

---

## Receiver (Rx) Specs

---

### Receiver Impedance

Receivers are required to meet the  $RL_{RX-DIFF}$  and  $RL_{RX-CM}$  (see Table 13-5 on page 498) parameters unless the device is powered down, as it would be, for example, in the L2 and L3 power states or during a Fundamental Reset. In those cases, a Receiver goes to the high impedance state and must meet the  $Z_{RX-HIGH-IMP-DC-NEG}$  and  $Z_{RX-HIGH-IMP-DC-NEG}$  parameters.

(See Table 13-5 on page 498.)

---

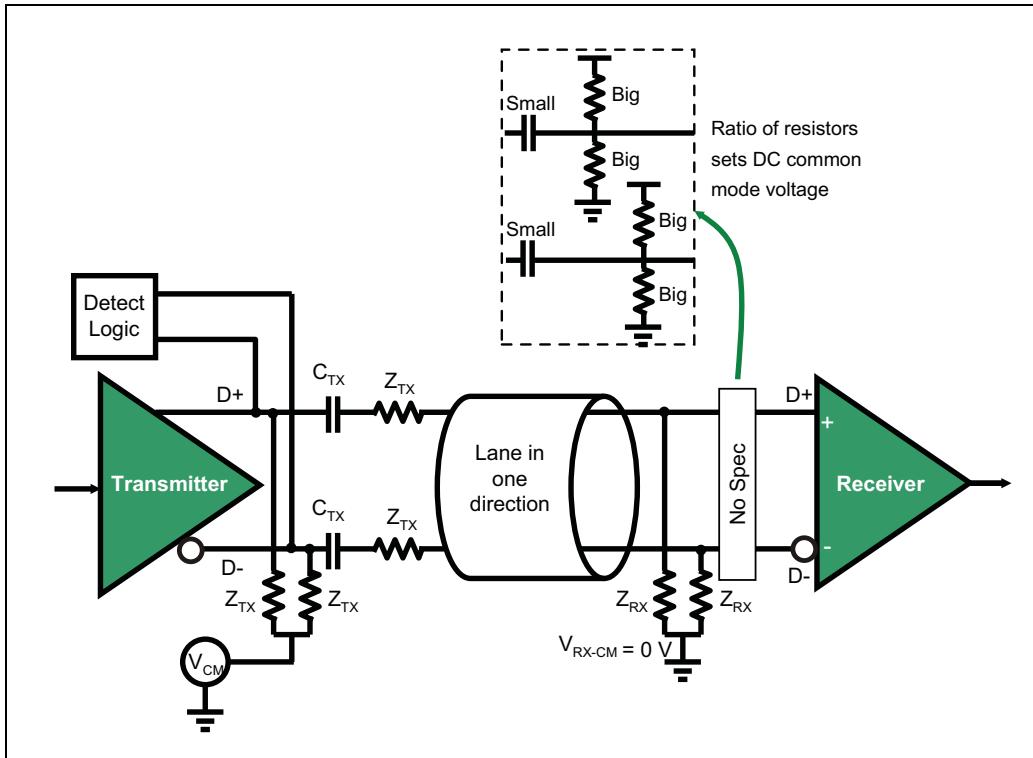
### Receiver DC Common Mode Voltage

The Receiver's DC common mode voltage is specified to be 0V for all data rates, and that's represented in Figure 13-15 on page 467 by showing the signal terminations connected to ground. The  $C_{TX}$  in-line capacitor permits this voltage to be something different at the Transmitter, which is specified to be in the range from 0 - 3.6V. That's not as interesting when the Transmitter and Receiver are in the same enclosure and have the same power supply, but if they're connected over a cable and reside in different machines with different power supplies it becomes more important. In that case it's difficult to avoid reference voltage differences between the machines and, since the signal voltages are already small, such a difference could make the signal difficult to recognize at the Receiver. The location of this capacitor must be near the Transmitter pins when a connector of some kind will be used but, if there's no connector, it can be located at any convenient place on the transmission line. Although it could be integrated into a device, it's expected that  $C_{TX}$  will be external because it would be too big to integrate.

## Chapter 13: Physical Layer - Electrical

The drawing in Figure 13-15 on page 467 also shows an optional set of resistors at the Receiver, labeled as “No Spec” because they are not mentioned in the spec. The story here is that Receiver designers dislike using a common-mode voltage of zero for the simple reason that it usually requires them to implement two reference voltages, one above zero and one below it. A preferred implementation offsets the signal entirely above or below zero, so that only one reference voltage is needed. The circuit shown within the dotted line accomplishes this by adding a small-value in-line capacitor to de-couple the DC component of the signal on the wire from that of the Receiver itself. Then, a resistor ladder serves to offset the Receiver’s common-mode voltage in one direction or the other to accomplish the goal.

Figure 13-15: Receiver DC Common-Mode Voltage Adjustment



## Transmission Loss

The Transmitter drives a minimum differential peak-to-peak voltage  $V_{TX-DIFFp-p}$  of 800 mV. The Receiver sensitivity is designed for a minimum differential peak-to-peak voltage ( $V_{RX-DIFFp-p}$ ) of 175 mV. This translates to a 13.2dB loss budget that a Link is designed for. Although a board designer can determine the attenuation loss budget of a Link plotted against various frequencies, the Transmitter and Receiver eye diagram measurement are the ultimate determinant of loss budget for a Link. Eye diagrams are described in “Eye Diagram” on page 485. A Transmitter that drives up to the maximum allowed differential peak-to-peak voltage of 1200 mV can compensate for a lossy Link that has worst-case attenuation characteristics.

## AC Coupling

PCI Express requires in-line AC-coupling capacitors be placed on each Lane, usually near the Transmitter. The capacitors can be integrated onto the system board, or integrated into the device itself, although the large size they would need makes that unlikely. An add-in card with a PCI Express device on it must place the capacitors on the card close to the Transmitter or integrate the capacitors into the PCIe silicon. These capacitors provide DC isolation between two devices on both ends of a Link thus simplifying device design by allowing devices to use independent power and ground planes.

---

## Signal Compensation

### De-emphasis Associated with Gen1 and Gen2 PCIe

For 2.5 GT/s and 5.0 GT/s transmission, PCIe mandates the use of a fairly simple form of Transmitter equalization called **de-emphasis** to reduce the effects of signal distortion along the Link transmission line. This distortion problem is always present but gets worse with increased frequency and lossy transmission lines.

#### The Problem

As data rates get higher, the Unit Interval (UI - bit time) becomes smaller, with the result that it's increasingly difficult to avoid having the value in one bit time affect the value in another bit time. The channel always resists changes to the voltage level, The faster we attempt to switch voltage, the more pronounced

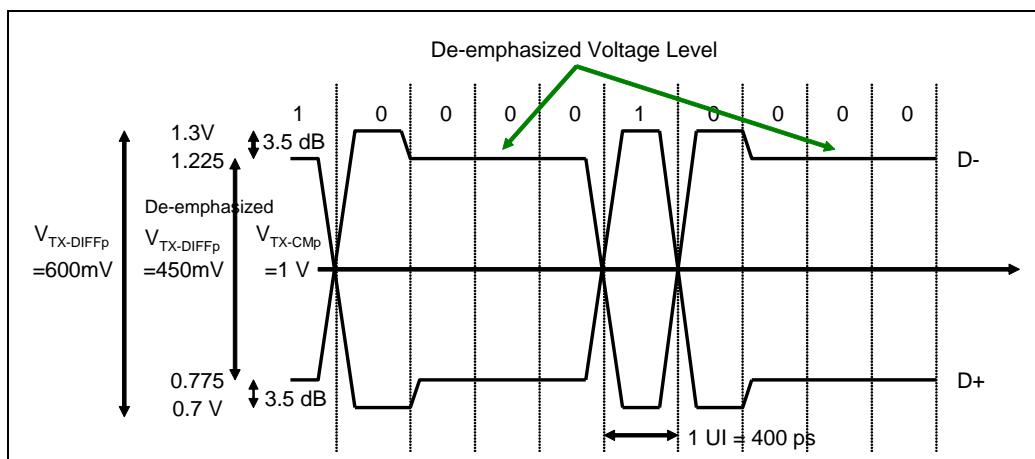
that effect becomes. However, when a signal has been held at the same voltage for several bit times, as when sending several bits in a row of the same polarity, the channel has more time to approach the target voltage. The resulting higher voltage makes it difficult to change to the opposite value within the required time when the polarity does change. This problem of previous bits affecting subsequent bits is referred to as ISI (inter-symbol interference).

## How Does De-Emphasis Help?

De-emphasis reduces the voltage for repeated bits in a bit stream. Although it sounds counter-intuitive at first because this reduces the signal swing and thus the energy that reaches the Receiver, reducing the Transmitter voltage for these cases can substantially improve signal quality. Figure 13-16 on page 469 illustrates how this works by showing a Transmitter output of '1000010000', where the repeated bits of the same polarity have been de-emphasized. De-emphasis can be thought of as a two-tap Tx equalizer, and some rules related to it are:

- When the signal changes to the opposite polarity of the preceding bit it's not de-emphasized, but uses the peak-to-peak differential voltage as specified by  $V_{TX-DIFFp-p}$  (see Table 13-3 on page 489).
- The first bit of a series of same polarity bits is not de-emphasized.
- Only subsequent bits of the same polarity after the first bit are de-emphasized.
- The de-emphasized voltage is reduced by 3.5 dB from normal for 2.5 GT/s, which translates to about a one-third reduction in voltage.
- The Beacon signal is de-emphasized, too, but uses slightly different rules. (see "Beacon Signaling" on page 483).

*Figure 13-16: Transmission with De-emphasis*



## Solution for 2.5 GT/s

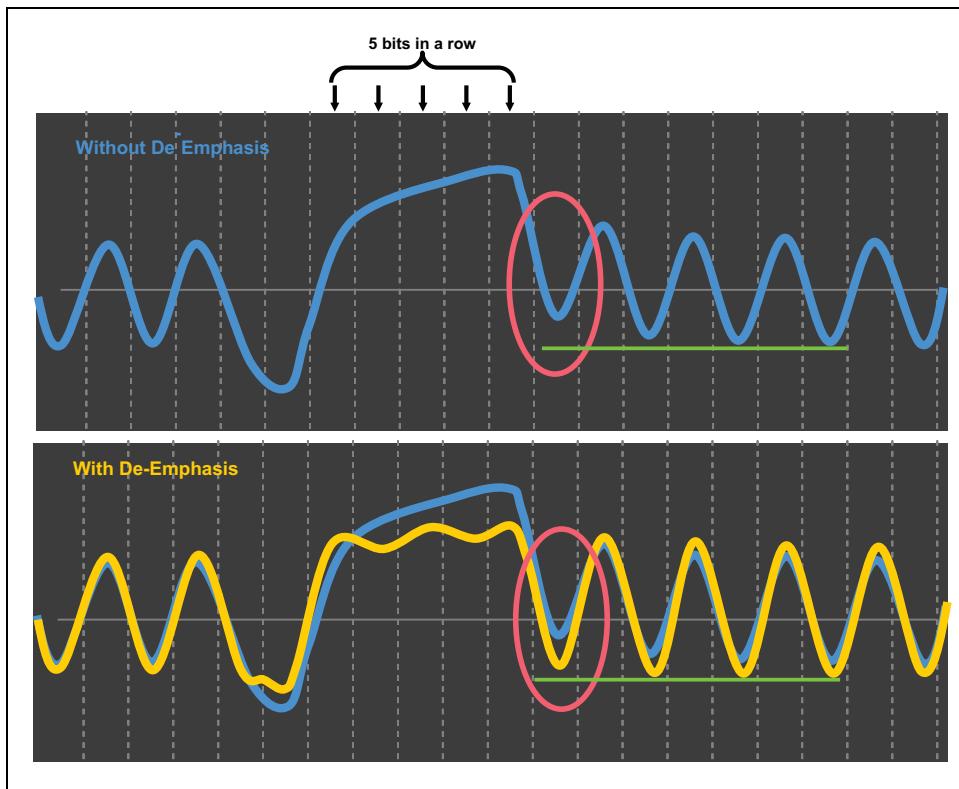
For 2.5 GT/s, each subsequent bit transmitted after the first bit of the same polarity must be de-emphasized by 3.5dB to accommodate this worst-case loss budget. Of course, for low-loss environments this is less important and for a very short path it can even make the received signal look worse. After all, de-emphasis is essentially distorting the transmitted signal in the opposite way of the distortion that is expected during transmission so as to cancel it out. If there turns out to be little or no distortion, then de-emphasis will make the signal look worse. The spec doesn't describe any way to test the signal environment or adjust the de-emphasis level, but doesn't prohibit a designer from developing an implementation-specific method of doing so.

An example of the benefit of de-emphasis is shown in Figure 13-17 on page 471, which is a scope capture converted into a drawing for clarity. The captures were taken from a device driving a long path and using a bit stream with several repeated bits to show the signal distortion. The trace at the top shows that the bit pattern for one side of the differential pair (also called a single-ended signal) has 2 bits of one polarity followed by 5 bits of the opposite polarity. Five consecutive bits is the worst case for 8b/10b, and this particular pattern only appears in a few characters like the COM character. The channel resists high-speed changes but will continue to charge up if the driver keeps trying to reach a higher voltage and that can be seen in this example. When the bits aren't repeated there isn't time for the voltage to go as far, but repeated bits give more time for the change. The problem this creates is seen in the bit following the 5<sup>th</sup> in a row (highlighted in the oval), which fails to reach a good signal value during its UI because the voltage difference was too large to overcome in that short time. The difference between the value it reaches and the value it should have reached is shown by the line marking the level reached by other bits that aren't experiencing as much ISI.

In the lower half of the illustration, a de-emphasized version of the signal is captured and compared to the original. Here we can see that reducing the voltage for repeated bits prevents the voltage from charging up as much and results in a cleaner signal because the bits that follow are not influenced as much by the previous bits. For both the 2 consecutive bits and then the 5 consecutive bits, the over-charging problem is reduced, which improves the timing jitter as well as the voltage levels. Consequently, the troublesome bit looks much better with de-emphasis turned on and the received signal approaches the normal voltage swing in that bit time.

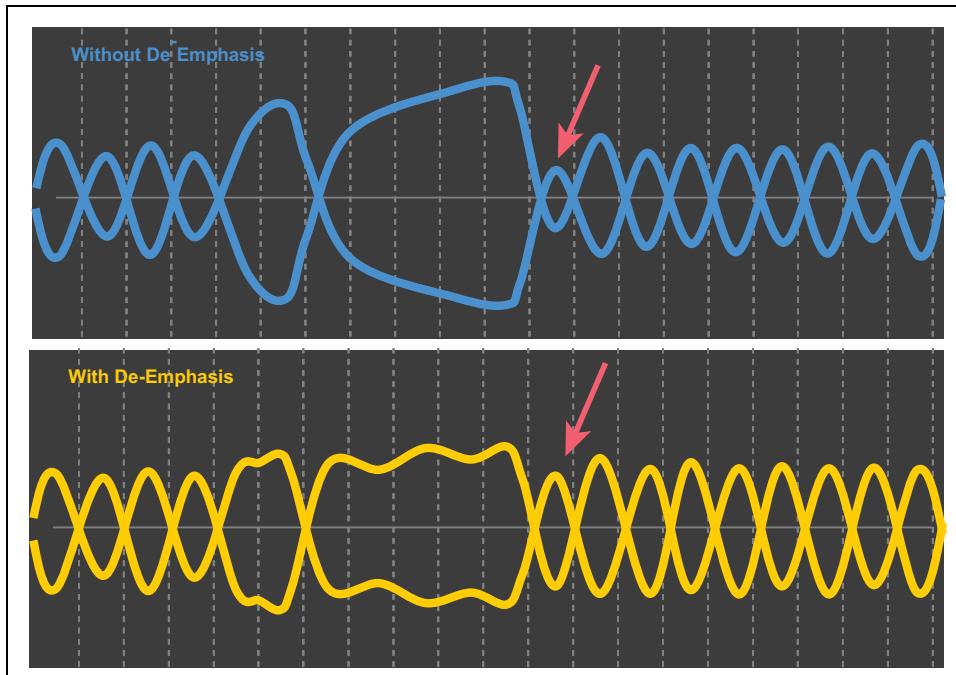
## Chapter 13: Physical Layer - Electrical

Figure 13-17: Benefit of De-emphasis at the Receiver



In Figure 13-18 on page 472 both positive and negative versions of the differential signal are shown so as to illustrate the resulting eye opening. The improved signal quality from de-emphasis is clear because the eye opening at the troublesome time in the lower trace is so much larger than the one without de-emphasis in the upper trace.

Figure 13-18: Benefit of De-emphasis at Receiver Shown With Differential Signals



## Solution for 5.0 GT/s

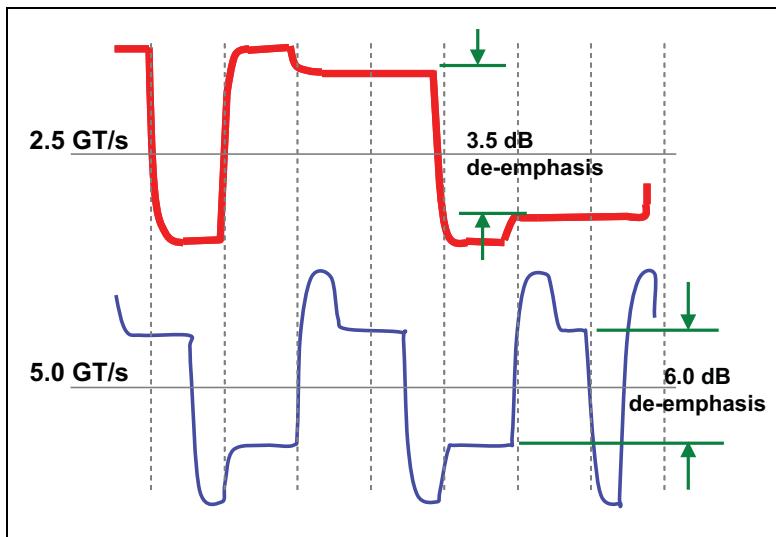
As one might expect, increasing data rates exacerbates the problem of ISI because the bit times get progressively smaller, and more aggressive equalization techniques are needed. The change for 5.0 GT/s is incremental, and consists of providing three choices regarding the amount of de-emphasis to be applied.

1. When running at 2.5 GT/s speed, -3.5 dB de-emphasis is required.
2. When running at 5.0 GT/s speed, -6.0 dB de-emphasis is recommended, while the use of -3.5 dB is optional. -6.0 dB de-emphasis level is intended to compensate for the greater signal attenuation at higher frequency. As Figure 13-19 on page 473 suggests, a 3.5 dB reduction represents a 33% reduction in voltage, while a 6 dB reduction represents a 50% reduction. To avoid a possible confusion, note that the dB measure of power and voltage are different by a factor of two. A 3 dB reduction represents a 50% change in power but only a 25% change in voltage.

## Chapter 13: Physical Layer - Electrical

---

Figure 13-19: De-emphasis Options for 5.0 GT/s

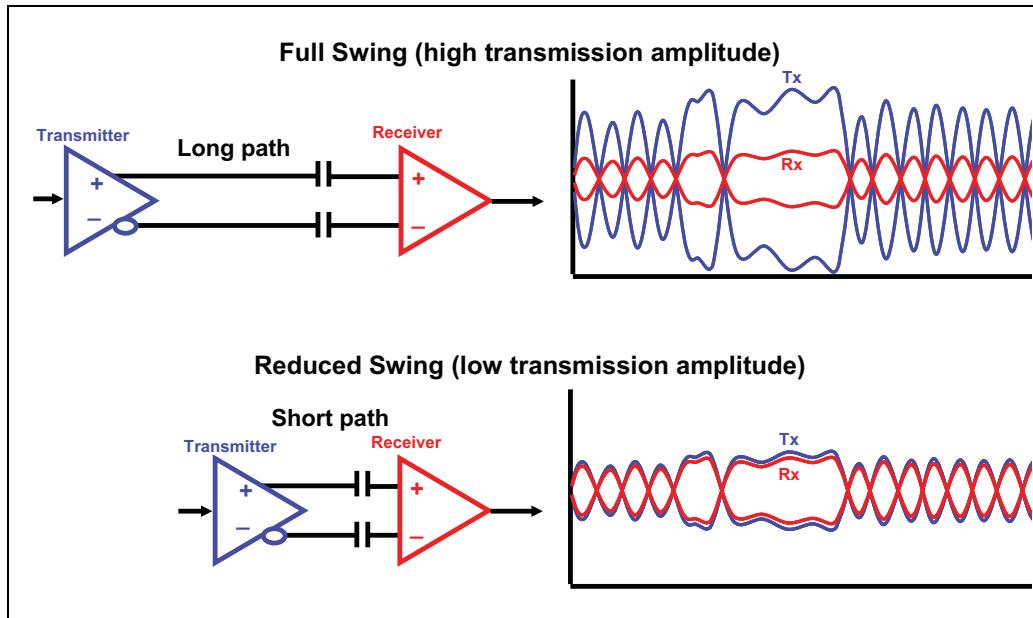


3. Normally, a Transmitter operates in the full-swing mode and can use the entire available voltage range to help overcome signal attenuation. The voltage needs to start out at a higher value to compensate for the loss, as shown in the top half of Figure 13-20 on page 474. However, for 5.0 GT/s another option is provided called reduced-swing mode. This is intended to support short, low-loss signaling environments, as shown in the lower half of Figure 13-20 on page 474, and reduces the voltage swing by about half to save power. This mode also provides the third de-emphasis option by turning off de-emphasis entirely, which makes sense because, as mentioned earlier, the signal distortion it creates would not be reduced by loss in the path and the resulting signal at the Receiver would look worse.

# PCI Express Technology

---

Figure 13-20: Reduced-Swing Option for 5.0 GT/s with No De-emphasis



---

## Solution for 8.0 GT/s - Transmitter Equalization

When going to the 8.0 GT/s data rate, the signal conditioning model changes significantly. Transmitter equalization becomes more complex and a handshake training procedure is used to adapt to the actual signaling environment rather than making assumptions about what will be needed. To learn more about the process of evaluating the Link, refer to the section called "Recovery.Equalization" on page 587. Basically, that process allows a Receiver to request that the Link partner's Transmitter use a certain combination of coefficients and then the receiver tests how well the received signal looks and possibly proposes others if the result isn't good enough.

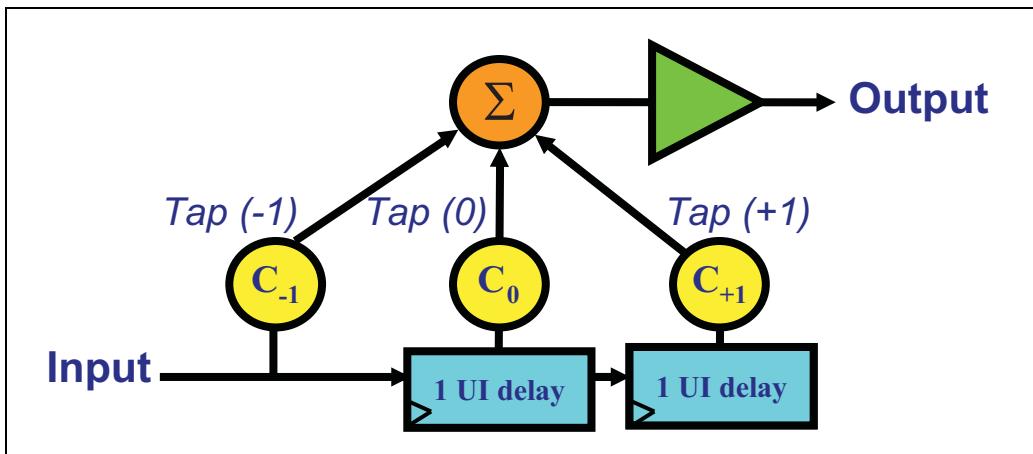
Sometimes students ask whether this model is really sufficient to achieve good error rates, since evaluating a signal across all the possible situations requires days of testing in the lab to achieve a BER of  $10^{-15}$  or better. The answer to this has two parts. First, even with the handshake process, the coefficients will be an approximation that worked well when the training was done but may or may not work as well under other conditions. Extrapolation from a small sample size

is a necessary part of arriving at working values quickly and it works reasonably well. Second, associated with 8 GT/s transfer rate, it's only necessary to achieve a minimum BER of  $10^{-12}$ , and that doesn't take as long to verify as it would BER of  $10^{-15}$ .

## Three-Tap Tx Equalizer Required

To accomplish better wave shaping at the Transmitter, the spec requires the use of a 3-tap FIR (Finite Impulse Response) filter, meaning a filter with 3 bit-time-spaced inputs. A conceptual drawing of this is shown in Figure 13-21 on page 475, where it can be seen that the output voltage is the sum of three versions of the input: the original input, a version delayed by one bit time and a third delayed by another bit time. This type of FIR filter is often used in other SERDES applications above 6.0 Gb/s, and it's helpful for PCIe because it compensates for the fact that the channel spreads the signal across a longer time. Another way of thinking about it is that a given bit is affected by both the bit value that preceded it and the bit that comes after it.

Figure 13-21: 3-Tap Tx Equalizer

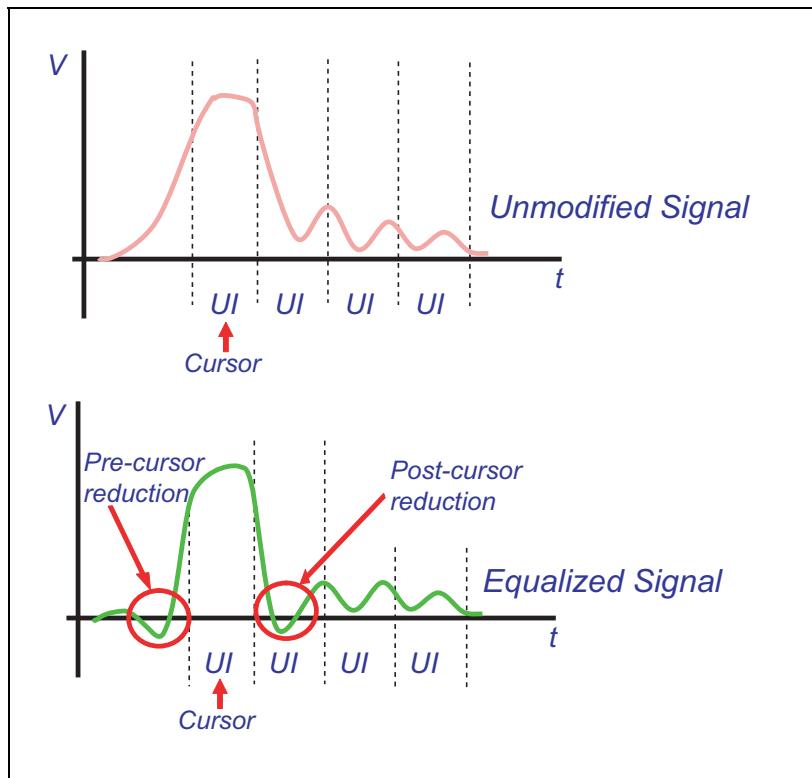


With this in mind, the three inputs can be described by their timing position as "pre-cursor" for  $C_{-1}$ , "cursor" for  $C_0$ , and "post-cursor" for  $C_{+1}$ , which combine to create an output based on the upcoming input, the current value, and the previous value. Adjusting the coefficients for the taps allows the output wave to be optimally shaped. This effect is illustrated by the pulse-response waveform shown in Figure 13-22 on page 476. Looking at a single pulse allows the adjustment to the signal to be more easily recognized.

# PCI Express Technology

The filter shapes the output according to the coefficient values (or tap weights) assigned to each tap. The sum of the absolute value of the three coefficient magnitudes together is defined to be unity so that only two of them need to be given for the third one to be calculated. Consequently, only  $C_{-1}$  and  $C_{+1}$  are given in the spec and  $C_0$  is always implied and is always positive.

Figure 13-22: Tx 3-Tap Equalizer Shaping of an Output Pulse



## Pre-shoot, De-emphasis, and Boost

The effect of the coefficient values is to adjust the output voltage to create up to four different voltage levels to accommodate different signaling environments, as shown in Figure 13-23 on page 477. This waveform was taken from a test device and shows a representative example, but the voltage levels depend on whether a Transmitter implements preshoot or de-emphasis or both.

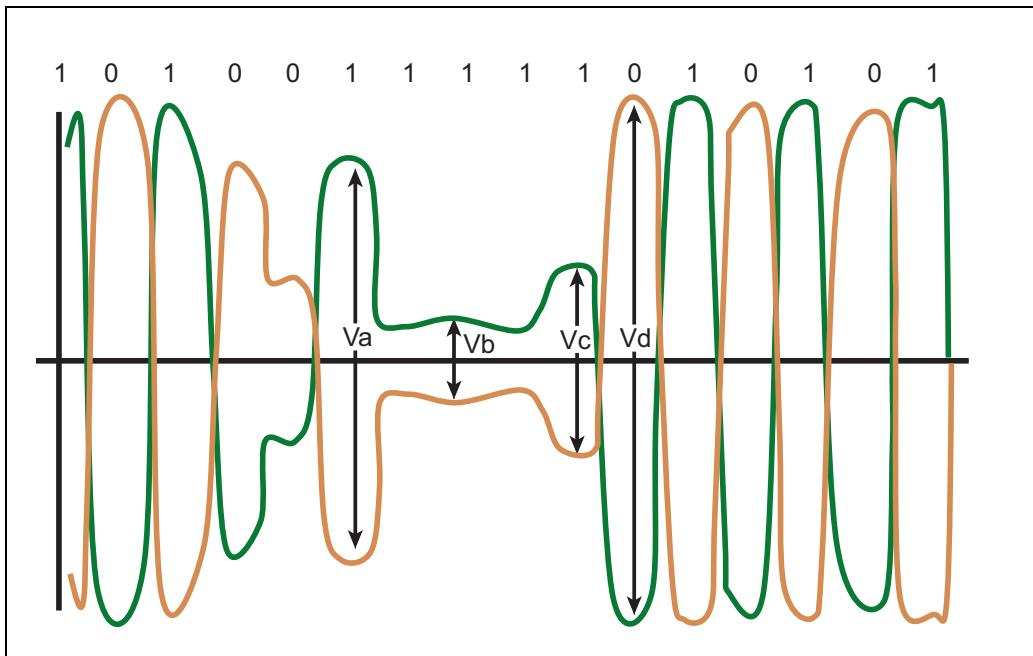
The waveform shows the four general voltages to be transmitted, which are: maximum-height ( $V_d$ ), normal ( $V_a$ ), de-emphasized ( $V_b$ ), and pre-shoot ( $V_c$ ).

## Chapter 13: Physical Layer - Electrical

This scheme is backward-compatible with the 2.5 and 5.0 GT/s model that only uses de-emphasis, because pre-shoot and de-emphasis can be defined independently. The voltages both with and without de-emphasis are the same as they have been for the lower data rates, except that now there are more options for the de-emphasis value, ranging from 0 to -6 dB. Preshoot is a new feature designed to improve the signal in the following bit time by boosting the voltage in the current bit time. Finally, the maximum value is simply what the signal would be if both  $C_{-1}$  and  $C_{+1}$  were zero (and  $C_0$  was 1.0). As illustrated by the bit stream shown at the top of the diagram, we may summarize the strategy for these voltages as follows:

- When the bits on both sides of the cursor have the opposite polarity, the voltage will be  $V_d$ , the maximum voltage.
- When a repeated string of bits is to be sent:
  - The first bit will use  $V_a$ , the next lower voltage to the maximum voltage  $V_d$ .
  - Bits between the first and last bits use  $V_b$ , the lowest voltage.
  - The last repeated bit before a polarity change uses  $V_c$ , the next higher voltage to the lowest voltage  $V_b$ .

Figure 13-23: 8.0 GT/s Tx Voltage Levels



## Presets and Ratios

As described in “Recovery.Equalization” on page 587, when the Link is preparing to change from a lower data rate to 8.0 GT/s, the Downstream Port sends EQ TS2s that give the Upstream Port a set of preset values to use for its coefficients as a starting point from which to begin testing the Link signal quality. The list of 11 possible presets along with their corresponding coefficient values and voltage ratios is given in Table 13-1 on page 478. Note that the voltages are given as a ratio with respect to the max value. These values were selected to match the earlier spec versions. As an example of how that is used, the first entry, P4, uses no de-emphasis or preshoot, so all the voltage values are equal to the max value and the ratios are all 1.000.

Table 13-1: Tx Preset Encodings with Coefficients and Voltage Ratios

Preset Number	Preshoot (dB)	De-emphasis (dB)	C <sub>-1</sub>	C <sub>+1</sub>	V <sub>a</sub> /V <sub>d</sub>	V <sub>b</sub> /V <sub>d</sub>	V <sub>c</sub> /V <sub>d</sub>
P4	0.0.	0.0	0.000	0.000	1.000	1.000	1.000
P1	0.0.	-3.5 +/- 1 dB	0.000	-0.167	1.000	0.668	0.668
P0	0.0.	-6.0 +/- 1.5 dB	0.000	-0.250	1.000	0.500	0.500
P9	3.5 +/- 1 dB	0.0	-0.166	0.000	0.668	0.668	1.000
P8	3.5 +/- 1 dB	-3.5 +/- 1 dB	-0.125	-0.125	0.750	0.500	0.750
P7	3.5 +/- 1 dB	-6.0 +/- 1.5 dB	-0.100	-0.200	0.800	0.400	0.600
P5	1.9 +/- 1 dB	0.0	-0.100	0.000	0.800	0.800	1.000
P6	2.5 +/- 1 dB	0.0	-0.125	0.000	0.750	0.750	1.000
P3	0.0	-2.5 +/- 1 dB	0.000	-0.125	1.000	0.750	0.750
P2	0.0	-4.4 +/- 1.5 dB	0.000	-0.200	1.000	0.600	0.600
P10	0.0	Defined by LF	0.000	(FS-LF)/2	1.000	Not fixed	Not fixed

## Equalizer Coefficients

Presets allow a device to use one of 11 possible starting values to be used for the partner's Transmitter coefficients when first training to the 8.0 GT/s data rate. This is accomplished by sending EQ TS1s and EQ TS2s during training which gives a coarse adjustment of Tx equalization as a starting point. If the signal using the preset delivers the desired  $10^{-12}$  error rate, no further training is needed. But if the measured error rate is too high, the equalization sequence is used to fine-tune the coefficient settings by trying different  $C_{-1}$  and  $C_{+1}$  values and evaluating the result, repeating the sequence until the desired signal quality or error rate is achieved.

An 8.0 GT/s transmitter is required to report its range of supported coefficient values to its neighboring Receiver. There are some constraints on this:

- Device must support all 11 presets as listed in Table 13-1 on page 478.
- Transmitters must meet the full-swing  $V_{TX-EIEOS-FS}$  signaling limits
- Transmitters may optionally support the reduced-swing, and if they do they must meet the  $V_{TX-EIEOS-RS}$  limits
- Coefficients must meet the boost limits ( $V_{TX-BOOST-FS} = 8.0$  dB min,  $V_{TX-BOOST-RS} = 2.5$  dB min) and resolution limits ( $EQ_{TX-DOEFF-RESS} = 1/24$  max to  $1/63$  min).

Applying these constraints and using the maximum granularity of 1/24 creates a list of pre-shoot, de-emphasis, and boost values for each setting. This is presented in a table in the spec that is partially reproduced from the spec here in Table 13-2 on page 480. The table contains blank entries because the boost value can't exceed  $8.0 \pm 1.5$  dB = 9.5 dB. That results in a diagonal boundary where the boost has reached 9.5 for the full-swing case. For reduced swing, the boundary is at 3.5 dB. The 6 shaded entries along the left and top edges of the table that go as far as 4/24 are presets supported by full- or reduced-swing signaling. The other 4 shaded entries are presets supported for full-swing signaling only.

# PCI Express Technology

---

Table 13-2: Tx Coefficient Table

PS DE Boost		C <sub>+1</sub>						
		0/24	1/24	2/24	3/24	4/24	5/24	6/24
C <sub>1</sub>	0/24	0.0 0.0 0.0	0.0 -0.8 0.8	0.0 -1.8 1.6	0.0 -2.5 2.5	0.0 -3.5 3.5	0.0 -4.7 4.7	0.0 -6.0- 6.0
	1/24	0.8 0.0 0.8	0.8 -0.8 1.6	0.9 -1.7 2.5	1.0 -2.8 3.5	1.2 -3.9 4.7	1.3 -5.3 6.0	1.6 -6.8 7.6
	2/24	1.6 0.0 1.6	1.7 -0.9 2.5	1.9 -1.9 3.5	2.2 -3.1 4.7	2.5 -4.4 6.0	2.9 -6.0 7.6	3.5 -8.0 9.5
	3/24	2.5 0.0 2.5	2.8 -1.0 3.5	3.1 -2.2 4.7	3.5 -3.5 6.0	4.1 -5.1 7.6	4.9 -7.0 9.5	-
	4/24	3.5 0.0 3.5	3.9 -1.2 4.7	4.4 -2.5 6.0	5.1 -4.1 7.6	6.0 -6.0 9.5	-	-
	5/24	4.7 0.0 4.7	5.3 -1.3 6.0	6.0 -2.9 7.6	7.0 -4.9 9.5	-	-	-
	6/24	6.0 0.0 6.0	6.8 -1.6 7.6	8.0 -3.5 9.5	-	-	-	-

**Coefficient Example.** Let's drill a little deeper on the coefficients by using preset number P7 from Table 13-1 on page 478 as an example. In this entry, C<sub>-1</sub> = -0.100, and C<sub>+1</sub> = -0.200, and since C<sub>0</sub> must be positive and the sum of their absolute values must be one, it's implied that C<sub>0</sub> = 0.700.

Matching these values to the table of coefficient space given in the spec is not straightforward because the coefficients are given as fractions rather than decimal values, but converting the fractions to their decimal values matches them pretty closely. The C<sub>-1</sub> value of 0.100 is closest to 2/24 (0.083), while C<sub>+1</sub> at 0.200 is a little less than 5/24 (0.208). The coefficient table entry for those fractions is highlighted as one of the preset values, giving us some confidence that this is on the right track. In the preset table, P7 lists a preshoot value of 3.5 +/- 1 dB, and the value in the coefficient table is shown as 2.9 dB. If we correct for the difference in coefficient values, ((0.083/.1) \* 3.5 = 2.9) we arrive at the same preshoot value. The difference in coefficient values for de-emphasis was much smaller (0.200 vs. 0.208) and so, as we might expect, both tables show this as -6.0 dB.

## Chapter 13: Physical Layer - Electrical

---

What voltages do the P7 coefficients create? Assuming a full-swing voltage of Vd as a starting point then, according to the ratios in the preset table, the other voltages would be Va = 0.8Vd, Vb = 0.4Vd, and Vc = 0.6Vd. How well do those correspond to the values that would result from using the pre-shoot and de-emphasis numbers? De-emphasis was given as -6.0 dB, and we already know that represents a 50% voltage reduction, so we'd expect that Vb should be half of Va, which it is. Pre-shoot was given as 3.5 dB meaning the ratio of Vc/Vb is 0.668, and  $0.4/0.668 = 0.598Vd$  for Vc; very close to the 0.6Vd we expected. Last of all, the Boost value, which is the ratio of Vd/Vb, is not given in the preset table but, using the formula  $20*\log(Vd/Vb)$ , the boost from the preset values turns out to be 7.9 dB. That's reasonably close to the 7.6 dB value given in the coefficient table and gives us some confidence that the tables are consistent among themselves.

So how are the four voltages obtained? There are essentially three programmable drivers whose output is summed to derive the final signal value to be launched. If the cursor setting remains unchanged, and the pre- and post-cursor taps are negative, then the answer can be found by simply adding the taps as  $(C_0 + C_{-1} + C_{+1})$ .

- $Vd = (C_0 + C_{-1} + C_{+1}) = (0.700 + 0.100 + 0.200) = 1.0 * \text{max voltage}$ . This is the “boosted” value that results when a bit is both preceded and followed by bits of the opposite polarity. In all four voltages listed here, if the polarity of the bits is inverted then the values would all be negative.
- $Va = (0.700 + (-0.100) + 0.200) = 0.8 * \text{max voltage}$ . This is the value that results when a bit is preceded by the opposite polarity but followed by the same polarity, meaning it is the first in a repeated string of bits.
- $Vb = (0.700 + (-0.100) + (-0.200)) = 0.4 * \text{max voltage}$ . This is the de-emphasized value that results when a bit is both preceded and followed by bits of the same polarity, meaning it's in the middle of a repeated string of bits.
- $Vc = (0.700 + 0.100 + (-0.200)) = 0.6 * \text{max voltage}$ . This is the pre-shoot value that results when a bit is preceded by the same polarity but followed by the opposite polarity, meaning it's the last bit in a repeated string of bits.

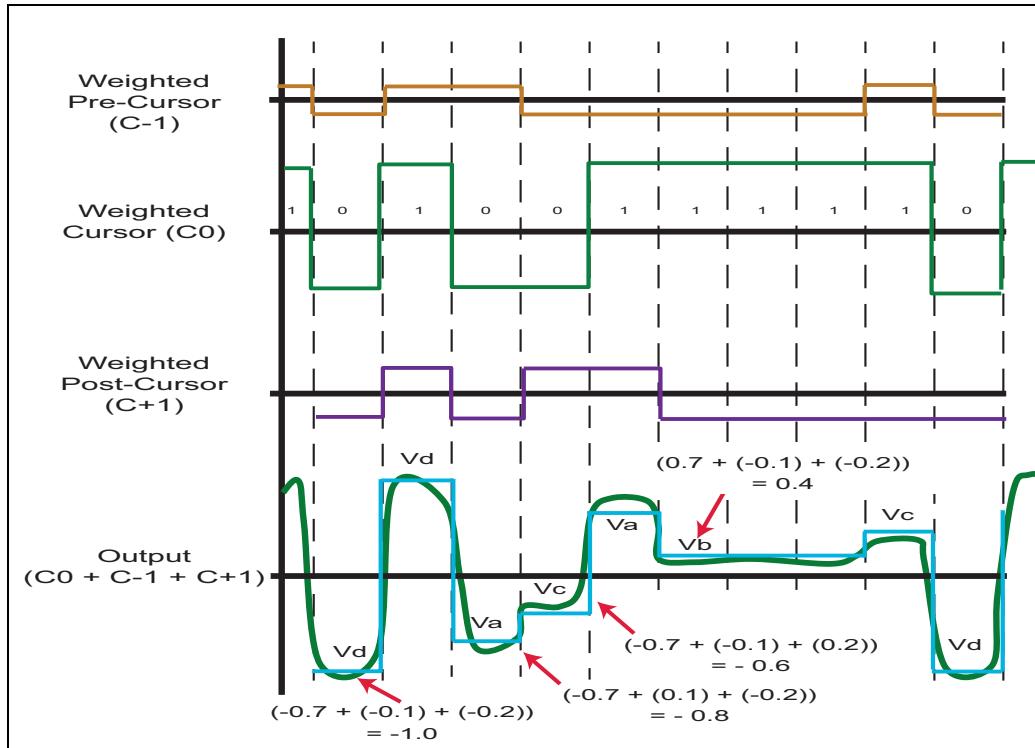
What determines when the coefficients are added or subtracted to arrive at these numbers? This turns out to be fairly simple, since it's just a matter of the polarity of the time-shifted pre- and post-cursor inputs. This is illustrated in Figure 13-24 on page 482. The single-ended waveform labeled “Weighted Cursor ( $C_0$ )” shows the positive half of the differential bit stream currently being transmitted. If the waveforms are understood as shifting to the right with time, then the next lower trace ( $C_{+1}$ ) is the post-cursor signal.

# PCI Express Technology

This version arrives one clock later and is weighted negatively by its coefficient, causing it to be inverted. The top trace ( $C_{-1}$ ) arrives a clock earlier than the cursor and is the pre-cursor value that is also weighted negatively according to its own coefficient.

Finally, the bottom trace shows the result of summing all three inputs to arrive at the final signal that is actually launched onto the wire. In the illustration, this is overlaid with the single-ended output waveform from Figure 13-23 on page 477 to show that it approximates a real capture fairly well. Some voltage calculations are shown from our previous example to demonstrate how the resulting voltages are obtained.

Figure 13-24: Tx 3-Tap Equalizer Output



## **Chapter 13: Physical Layer - Electrical**

---

The coefficient presets are exchanged before the Link changes to 8.0 GT/s, and then they may be updated during the Link equalization process (see “Recovery.Equalization” on page 587 for more details).

**EIEOS Pattern.** At 8.0 GT/s, some voltages are measured when the signal has a low frequency because the high-frequency changes won’t reach the levels we want to measure. The EIEOS sequence contains 8 consecutive ones followed by 8 consecutive zeros in a pattern that repeats for 128 bit times. Its purpose is primarily to serve as an unambiguous indication that a Transmitter is exiting from Electrical Idle, which scrambled data can’t guarantee. Its launch voltage is defined as  $V_{TX-EIEOS-FS}$  for full swing and  $V_{TX-EIEOS-RS}$  for reduced swing signals.

**Reduced Swing.** Transmitters may support a reduced swing signal much as they did for 5.0 GT/s: to achieve both power savings and a better signal over short, low-loss transmission paths. The output voltage has the same 1300 mV max value as the full-swing case, but allows a lower minimum voltage of 232 mV as defined for  $V_{TX-EIEOS-RS}$ . Operating at reduced swing limits the number of presets because the maximum boost supported is 3.5 dB.

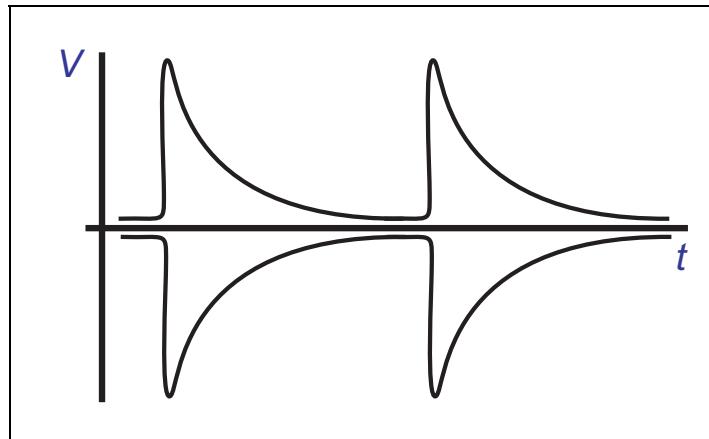
---

## **Beacon Signaling**

### **General**

De-emphasis is also applied to the Beacon signal, so a discussion about the Beacon is included in this section. A device whose Link is in the L2 state can generate a wake-up event to request that power be restored so it can communicate with the system. The Beacon is one of two methods available for this purpose. The other method is to assert the optional sideband WAKE# signal. An example of what the Beacon might look like is shown in Figure 13-25 on page 484. This version shows the differential signals pulsing and then decaying in opposite directions and is reminiscent of a flashing beacon light. Other options are available for the Beacon, but this one illustrates the concept well.

Figure 13-25: Example Beacon Signal



While a Link is in L2 power state, its main power source and clock are turned off but an auxiliary voltage source ( $V_{aux}$ ) keeps a small part of the device working, including the wake-up logic. To signal a wake-up event, a downstream device can drive the Beacon upstream to start the L2 exit sequence. A switch or bridge receiving a Beacon on its Downstream Port must forward notification upstream by sending the Beacon on its Upstream Port or by asserting the WAKE# pin. See "WAKE#" on page 773.

The motivation for creating two wake-up mechanisms is to provide choices regarding power consumption. To use the Beacon, all the bridges and switches between an Endpoint and the Root Complex will need to use  $V_{aux}$  so they can detect and generate the signal. If a system is always plugged in and unconcerned about the amount of standby power, the Beacon in-band signal may be preferred over having to route an extra side-band signal. But in a mobile system with limited battery life where conserving power is a high priority, the WAKE# pin is preferred because that approach uses as little  $V_{aux}$  as possible. The pin could be connected directly from the Endpoint to the Root Complex and then no other devices would need to be involved or use  $V_{aux}$ .

## Properties of the Beacon Signal

- A low-frequency, DC-balanced differential signal consisting of a periodic pulse of between 2ns and 16 $\mu$ s.
- The maximum time between pulses can be no more than 16 $\mu$ s.
- The transmitted Beacon signal must meet the electrical voltage specs documented in Table 13-3 on page 489.

# Chapter 13: Physical Layer - Electrical

---

- The signal must be DC balanced within a maximum time of 32 $\mu$ s.
- Beacon signaling, like normal differential signaling, must be done with the Transmitter in the low impedance mode (50  $\Omega$  single-ended, 100  $\Omega$  differential impedance).
- When signaled, the Beacon signal must be transmitted on Lane 0, but does not have to be transmitted on other Lanes.
- With one exception, the transmitted Beacon signal must be de-emphasized according to the rules defined in the previous section. For Beacon pulses greater than 500ns, the Beacon signal voltage must be 6db de-emphasized from the  $V_{TX-DIFFp-p}$  spec. The Beacon signal voltage may be de-emphasized by up to 3.5dB for Beacon pulses smaller than 500ns.

---

## Eye Diagram

---

### Jitter, Noise, and Signal Attenuation

As the bit stream travels from the Transmitter on one end of a link to the Receiver on the other end, it is subject to the following disruptive influences:

- Deterministic (i.e., predictable) jitter induced by the Link transmission line.
- Data-dependent jitter induced by the dynamic data patterns on the Link.
- Noise induced into the signal pair.
- Signal attenuation due to the impedance effect of the transmission line.

---

### The Eye Test

To verify that a Receiver sees an signal that is within the allowed variation, an eye test may be performed. The following description of this measurement was provided by James Edwards from an article he authored for *OE Magazine*.

*"The most common time-domain measurement for a transmission system is the eye diagram. The eye diagram is a plot of data points repetitively sampled from a pseudo-random bit sequence and displayed by an oscilloscope. The time window of observation is two data periods wide. For a [PCI Express link running at 2.5 GT/s], the period is 400ps, and the time window is set to 800ps. The oscilloscope sweep is triggered by every data clock pulse. An eye diagram allows the user to observe system performance on a single plot."*

*To observe every possible data combination, the oscilloscope must operate like a multiple-exposure camera. The digital oscilloscope's display persistence is set to infinite. With each clock trigger, a new waveform is measured and overlaid upon all*

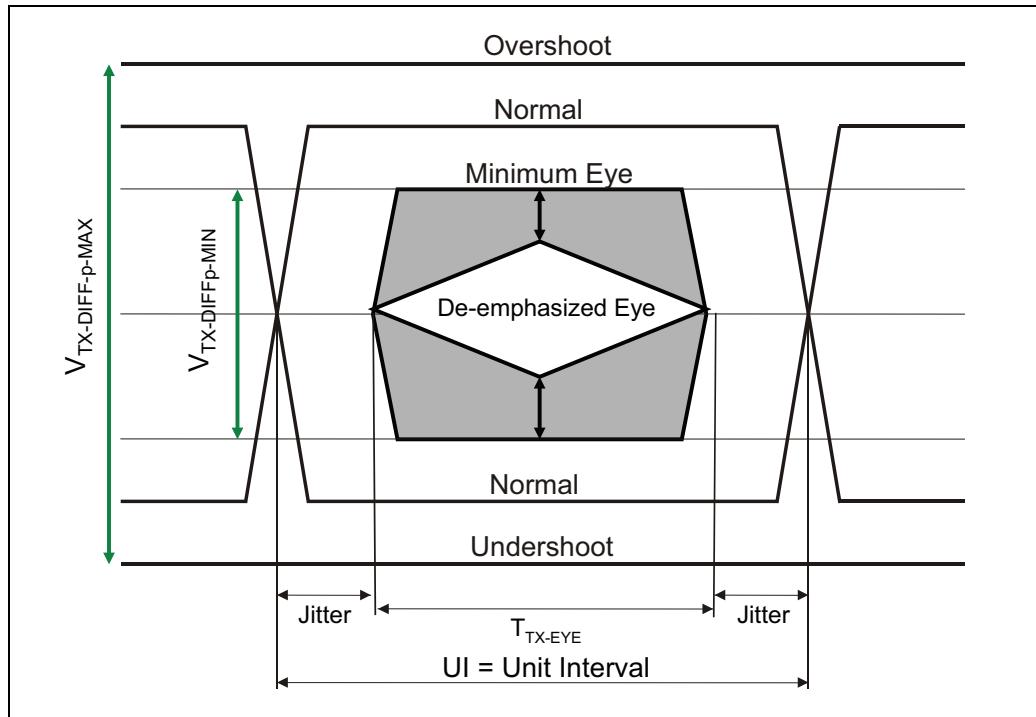
# PCI Express Technology

previous measured waveforms. To enhance the interpretation of the composite image, digital oscilloscopes can assign different colors to convey information on the number of occurrences of the waveforms that occupy the same pixel on the display, a process known as color-grading. Modern digital sampling oscilloscopes include the ability to make a large number of automated measurements to fully characterize the various eye parameters.”

## Normal Eye Diagram

An ideal trace capture would paint an eye pattern that matched the outline shown in the center of Figure 13-26 on page 486 labeled “Normal”. As long as the pattern resides entirely within that region, the Transmitter and Link are within tolerance. Note that the differential voltage parameters and values shown are peak voltages instead of the peak-to-peak voltages used in the spec, because only peak voltages can be represented in an eye diagram. Figure 13-27 on page 488 shows a screen capture of a good eye diagram.

Figure 13-26: Transmitter Eye Diagram



## Effects of Jitter

Jitter (timing uncertainty) is what happens when an edge arrives either before or after its ideal time, and acts to reduce signal integrity and close the eye opening. It's caused by a variety of factors, from environmental effects to the data pattern in flight, to noise or signal attenuation that causes the signal's voltage level to overshoot or undershoot the normal zone. At 2.5 GT/s this could be treated as a simple lumped effect, but at higher data rates it becomes a more significant issue and must be considered in several different parts. Aiming at this goal, the 8.0 GT/s data rate defines 5 different jitter values. The details of jitter analysis and minimization are beyond the scope of this book, but let's at least define the terms the spec uses. Jitter is described as being in one of several categories:

1. Un-correlated - jitter that is not dependent on, or "correlated" to, the data pattern being transmitted.
2.  $R_j$  - Random jitter from unpredictable sources that are unbounded and usually assumed to fit a Gaussian distribution. Often caused by electrical or thermal noise in the system.
3.  $D_j$  - Deterministic jitter that's predictable and bounded in its peak-to-peak value. Often caused by EMI, crosstalk, power supply noise or grounding problems.
4. PWJ - Pulse Width Jitter - uncorrelated, edge-to-edge, high-frequency jitter.
5.  $D_{jDD}$  - Deterministic Jitter, using the Dual-Dirac approximation. This model is a method of quickly estimating total jitter for a low BER without requiring the large sample size that would normally be needed. It uses a representative sample taken over a relatively short period (an hour or so) and extrapolates the curves to arrive at acceptable approximate values.
6.  $DD_j$  - Data-dependent jitter is a function of the data pattern being sent, and the spec states that this is mostly due to package loss and reflection. ISI is an example of  $DD_j$ .

Figure 13-28 on page 488 shows a screen capture of a bad Eye Diagram at 2.5 GT/s. Since this is captured without de-emphasis, the traces should all stay outside the Minimum Eye area, shown on the screen by the trapezoid shape in the middle. This example illustrates that jitter can affect both edge arrival times and voltage levels, causing some trace instances to encroach on the keep-out area of the diagram.

# PCI Express Technology

Figure 13-27: Rx Normal Eye (No De-emphasis)

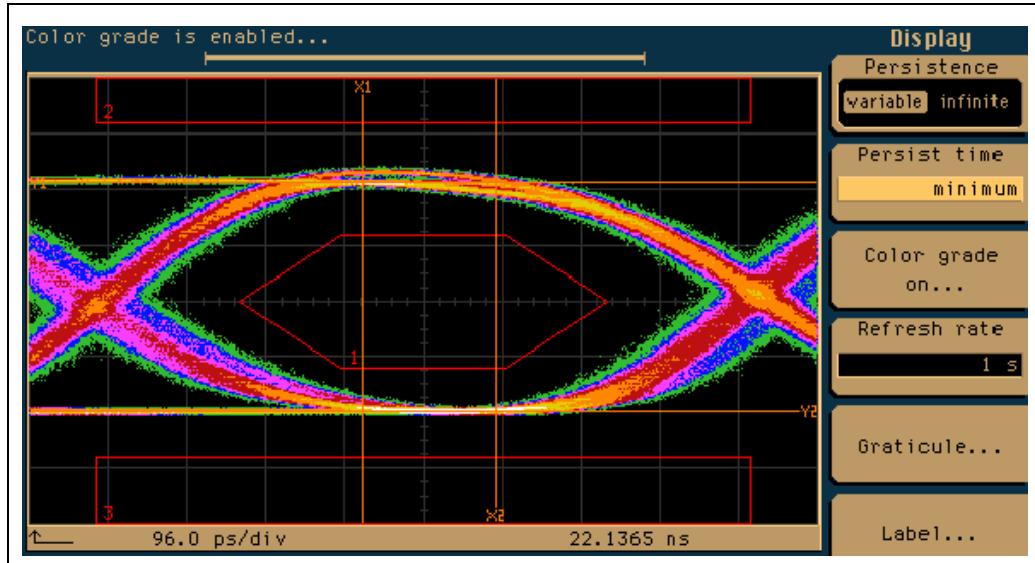
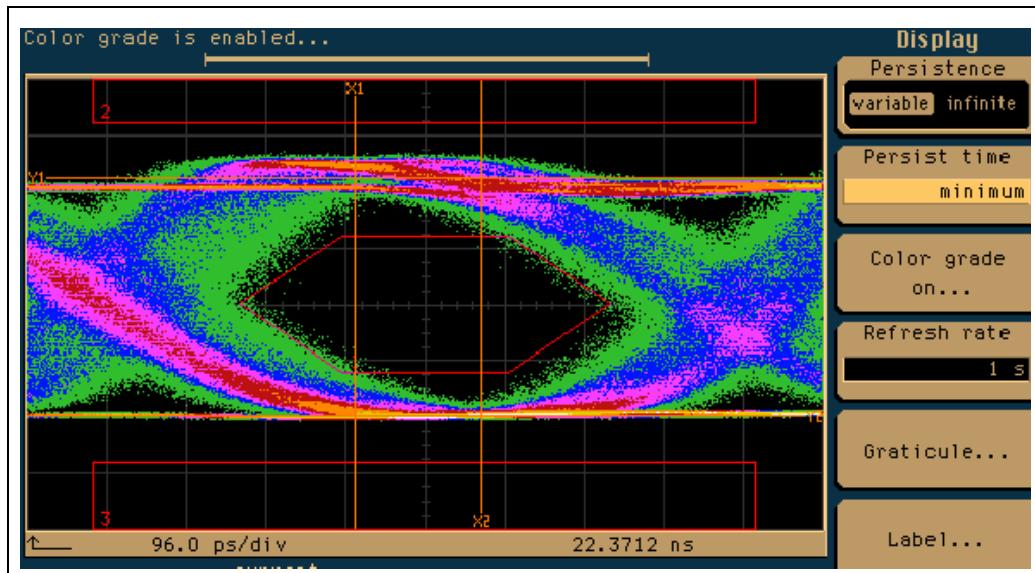


Figure 13-28: Rx Bad Eye (No De-emphasis)



# Chapter 13: Physical Layer - Electrical

---

## Transmitter Driver Characteristics

Table 13-3 on this page lists some Transmitter driver characteristics. This is not intended to replicate the tables from the spec, but to give some basic parameters to illustrate some differences between the data rates, such as UI, and to show that some things have remained unchanged, such as the Tx common-mode voltage.

*Table 13-3: Transmitter Specs*

Item	2.5 GT/s.	5.0 GT/s	8.0 GT/s	Units	Notes
UI	399.88 (min) 400.12 (max)	199.94 (min) 200.06 (max)	124.9625 (min) 125.0375 (max)	ps	Unit Interval (bit time)
T-TX-EYE	0.75 (min)	0.75 (min)	See notes	UI	Transmitter Eye, including all jitter sources. For 8.0 GT/s, five jitter sources are specified separately.
T-TX-RF-MISMATCH	Not Specified	0.1 (max)	Not Specified	UI	Rise and Fall time difference measured from 20% to 80% differentially.
V <sub>TX-DIFFp-p</sub>	0.8 (min) 1.2 (max)	0.8 (min) 1.2 (max)	See Table 13-4	mV	Peak-to-peak differential voltage.
V <sub>TX-DIFFp-p LOW</sub>	0.4 (min) 1.2 (max)	0.4 (min) 1.2 (max)	See Table 13-4	mV	Low-power voltage.
V <sub>TX-DC-CM</sub>	0 to 3.6	0 to 3.6	0 to 3.6	V	DC common mode voltage at Tx pins.
V <sub>TX-DE-RATIO-3.5dB</sub>	3 (min) 4 (max)	3 (min) 4 (max)	See Table 13-4	mV	Ratio for 3.5 dB de-emphasized bits.
V <sub>TX-DE-RATIO-6dB</sub>	n/a	5.5 (min) 6.5 (max)	See Table 13-4	mV	Ratio for 6 dB de-emphasized bits.

# PCI Express Technology

---

*Table 13-3: Transmitter Specs (Continued)*

Item	2.5 GT/s.	5.0 GT/s	8.0 GT/s	Units	Notes
I <sub>TX-SHORT</sub>	90	90	90	mA	Total single-ended current Tx can supply when shorted to ground.
V <sub>TX-IDLE-DIFF-AC-P</sub>	0 (min) 20 (max)	0 (min) 20 (max)	0 (min) 20 (max)	mV	Peak differential voltage under Electrical Idle state of Link. Must include a bandpass filter passing frequencies from 10 KHz to 1.25 GHz.
T <sub>TX-IDLE-MIN</sub>	20 (min)	20 (min)	20 (min)	ns	Minimum time a Transmitter must be in Electrical Idle.
T <sub>TX-IDLE-SET-TO-IDLE</sub>	8 (max)	8 (max)	8 (max)	ns	Time allowed for Tx to meet Electrical Idle spec after last bit of required EIOSs.
T <sub>TX-IDLE-TO-DIFF-DATA</sub>	8	8	8	ns	Max time for Tx to meet differential transmission spec after Electrical Idle exit.
Z <sub>TX-DIFF-DC</sub>	80 (min) 120 (max)	120 (max)	120 (max)	Ω	DC differential Tx impedance. Typical value is 100 Ω. Min value for 5.0 and 8.0 GT/s is bounded by R <sub>LTX-DIFF</sub>

## Chapter 13: Physical Layer - Electrical

---

*Table 13-3: Transmitter Specs (Continued)*

Item	2.5 GT/s.	5.0 GT/s	8.0 GT/s	Units	Notes
RL <sub>TX-DIFF</sub>	10 (min)	10 (min) for 0.5-1.25 GHz  8 (min) for >1.25-2.5 GHz	10 (min) for 0.5-1.25 GHz  8 (min) for >1.25 - 2.5 GHz  4 (min) for >2.5 to 4 GHz	dB	Tx package return loss. Note that the frequency is the signal on the wire. Note that at higher rates it becomes necessary to specify different parame- ters for different frequen- cies.
C <sub>TX</sub>	75 (min) 265 (max)	75 (min) 265 (max)	176 (min) 265 (max)	nF	Required AC coupling cap on each Lane placed in the media or in the component itself.
L <sub>TX-SKEW</sub>	500 ps + 2 UI (max)	500 ps + 4 UI (max)	500 ps + 6 UI	ps	Skew between any two Lanes in the same Trans- mitter.

*Table 13-4: Parameters Specific to 8.0 GT/s*

Symbol	Value	Units	Notes
V <sub>TX-FS-NO-EQ</sub>	1300 (max) 800 (min)	mvPP	No EQ is applied; measured using 64 zeros followed by 64 ones.
V <sub>TX-RS-NO-EQ</sub>	1300 (max)	mvPP	No EQ is applied; measured using 64 zeros followed by 64 ones.
V <sub>TX-BOOST-FS</sub>	8.0 (min)	dB	Tx boost ratio for full swing. (Assumes +/- 1.5 dB tolerance)
V <sub>TX-BOOST-RS</sub>	2.5 (min)	dB	Tx boost ratio for reduced swing. (Assumes +/- 1.0 dB tolerance)
EQ <sub>TX-COEFF-RES</sub>	1/24 (max) 1/63 (min)	n/a	Tx coefficient resolution

## Receiver Characteristics

---

### Stressed-Eye Testing

Receivers are tested using a stressed eye technique, in which a signal with specific problems is presented to the input pins and the BER is monitored. The spec presents these for 2.5 and 5.0 GT/s separately from 8.0 GT/s because of the difference in the methods used, and then gives a third section that defines parameters common to all the speeds.

#### 2.5 and 5.0 GT/s

At 2.5 GT/s, the parameters are measured at the Receiver pins and there is an implied correlation between the margins observed and the BER. At 5.0 GT/s, receiver tolerancing is applied. This is a two-step method in which a test board is calibrated to show the worst-case signal margins as defined in the spec. Then, once the calibration is done, the test load is replaced by the device to be tested and its BER is observed. There are actually two sets of worst-case numbers based on the clocking scheme: one is defined for the common-clock architecture and another for the data-clocked architecture. At higher speeds every element of the signal path must be carefully considered, and that's true for the device package, too. The effects added to the signal by the package must be comprehended in the testing process.

The calibration channel itself must be designed with specific characteristics in mind, but the spec observes that a trace length of 28 inches on an FR4 PCB should suffice to create the necessary ISI. A signal generator is used to inject the Compliance Pattern with the appropriate jitter elements included.

#### 8.0 GT/s

The method for testing the stressed eye at 8.0 GT/s is similar, but there are some differences. One difference is that the signal can't be evaluated at the device pin and so a replica channel is used to allow measuring the signal as it would appear at the pin if the device were an ideal termination.

In order to evaluate the Receiver's ability to perform equalization properly, it's recommended that multiple calibration channels with different insertion loss characteristics be used so the receiver can be tested in more than one environment. As with the transmitter at 8.0 GT/s, the calibration channel for the receiver consists of differential traces terminated at both ends with coaxial connectors.

# Chapter 13: Physical Layer - Electrical

---

To establish the correct correlation between the channel and the receiver it's necessary to model what the receiver see internally after equalization has been applied. That means post processing is must be applied that will model what happens in the Receiver, including the following items, the details of which are described in the spec:

- Package insertion loss
- CDR - Clock and Data Recovery logic
- Equalization that accounts for the longest calibration channel, including
  - First-order CTLE (Continuous Time Linear Equalizer)
  - One-tap DFE (Decision Feedback Equalizer)

---

## Receiver (Rx) Equalization

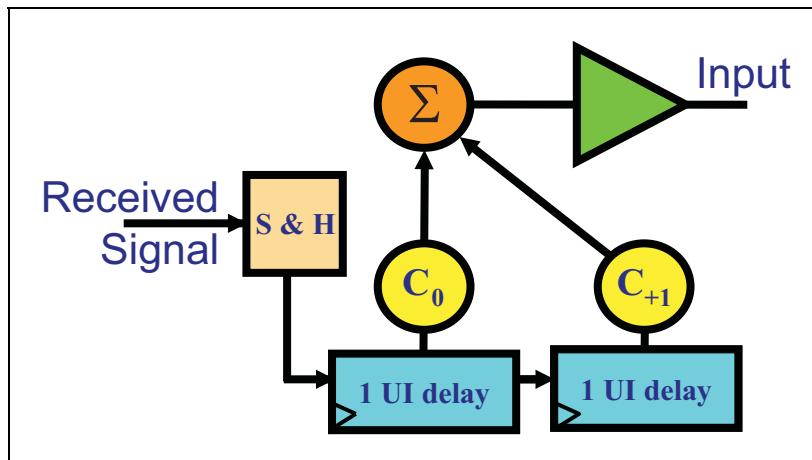
Transmitter equalization is mandatory, but the signal may still suffer enough degradation going through the longest permissible channel that the eye is closed and the signal is unrecognizable at the Receiver. To accommodate this the spec describes receiver equalization logic, but says is not intended to serve as an implementation guideline. What it does say is that a version will be required for calibrating the stressed eye when using the longest allowed calibration channel. As described earlier, that requirement is described as a first-order CTLE and a one-tap DFE.

### Continuous-Time Linear Equalization (CTLE)

A linear equalizer removes the undesirable frequency components from the received signal. For PCIe this could be as simple as a passive high-pass filter that reduces the voltage of the low frequency component from the received signal which attenuates by a lower amount on the transmission line. It could also be done with amplification to open up the received eye, however that would amplify the high-frequency noise along with the signal and create other problems.

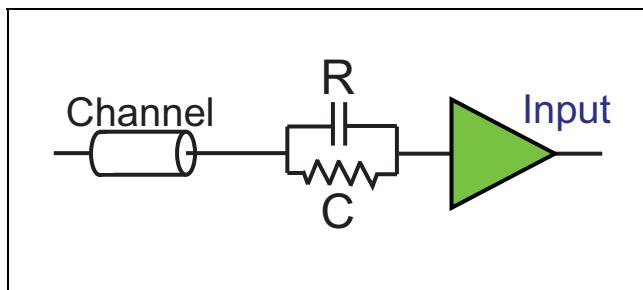
One form of receiver equalization would be a circuit like the one shown in Figure 13-29 on page 494, which is a Discrete Time Linear Equalizer (DLE). This is simply an FIR filter, similar to the one used by the transmitter, to provide wave shaping as a means of compensating for channel distortion. One difference is that it uses a Sample and Hold (S & H) circuit on the front end to hold the analog input voltage at a sampled value for a time period, rather than allowing it to constantly change. The spec doesn't mention DLE, and the reasons may include its higher cost and power compared to CTLE. As with the transmitter FIR, more taps provide better wave shaping but add cost, so only a small number are practical.

Figure 13-29: Rx Discrete-Time Linear Equalizer (DLE)



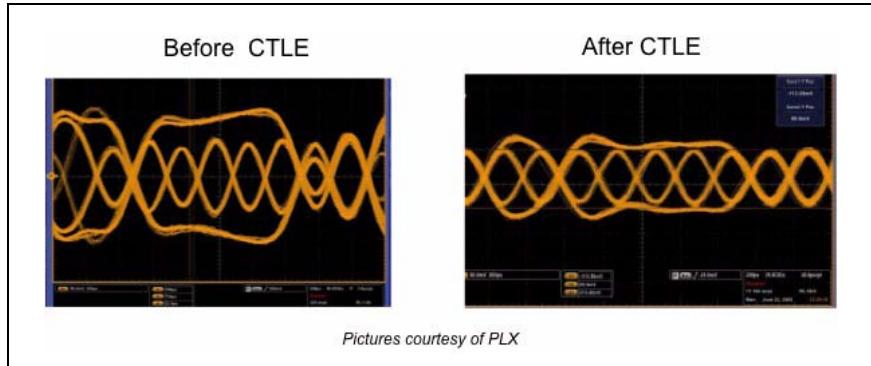
In contrast, CTLE is not limited to discrete time intervals and improves the signal over a longer time interval. A simple RC network can serve as an example of a CTLE high-pass filter, as shown in Figure 13-30 on page 494. This serves to reduce the low-frequency distortion caused by the channel without boosting the noise in the high-frequency range of interest and cleans the signal for use at the next stage. Figure 13-31 on page 495 illustrates the attenuation effect of CTLE high-pass filter on the received low frequency component of a signal e.g. continuous 1s or continuous 0s.

Figure 13-30: Rx Continuous-Time Linear Equalizer (CTLE)



# Chapter 13: Physical Layer - Electrical

Figure 13-31: Effect of Rx Continuous-Time Linear Equalizer (CTLE) on Received Signal

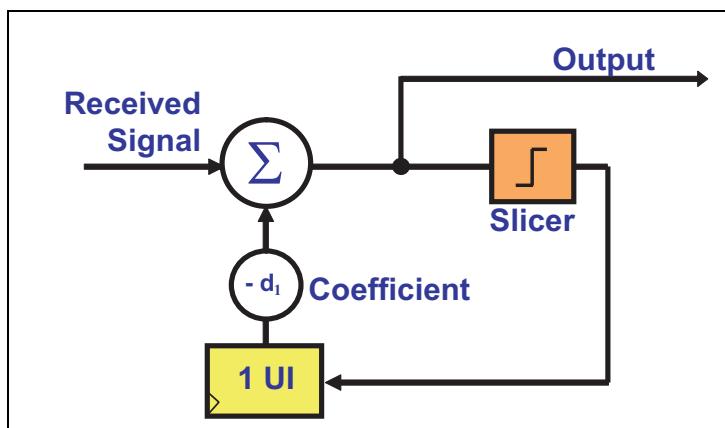


Pictures courtesy of PLX

## Decision Feedback Equalization (DFE)

An example one-tap DFE circuit like the one described in the spec is shown in Figure 13-32 on page 495, where it can be seen that the received signal is summed with the feedback value and then fed into a data "slicer." A slicer is an A/D circuit that takes the analog-looking input and converts it into a clean, full-swing digital signal for internal use. It makes its best guess and decides whether the input is a positive or negative value and outputs either +1 or -1. This decision is sent into an FIR filter with only one tap, which is just a delayed version weighted according to a coefficient setting. The output of this filter is then fed back and summed with the received signal for use as the new input to the data slicer.

Figure 13-32: Rx 1-Tap DFE



## PCI Express Technology

---

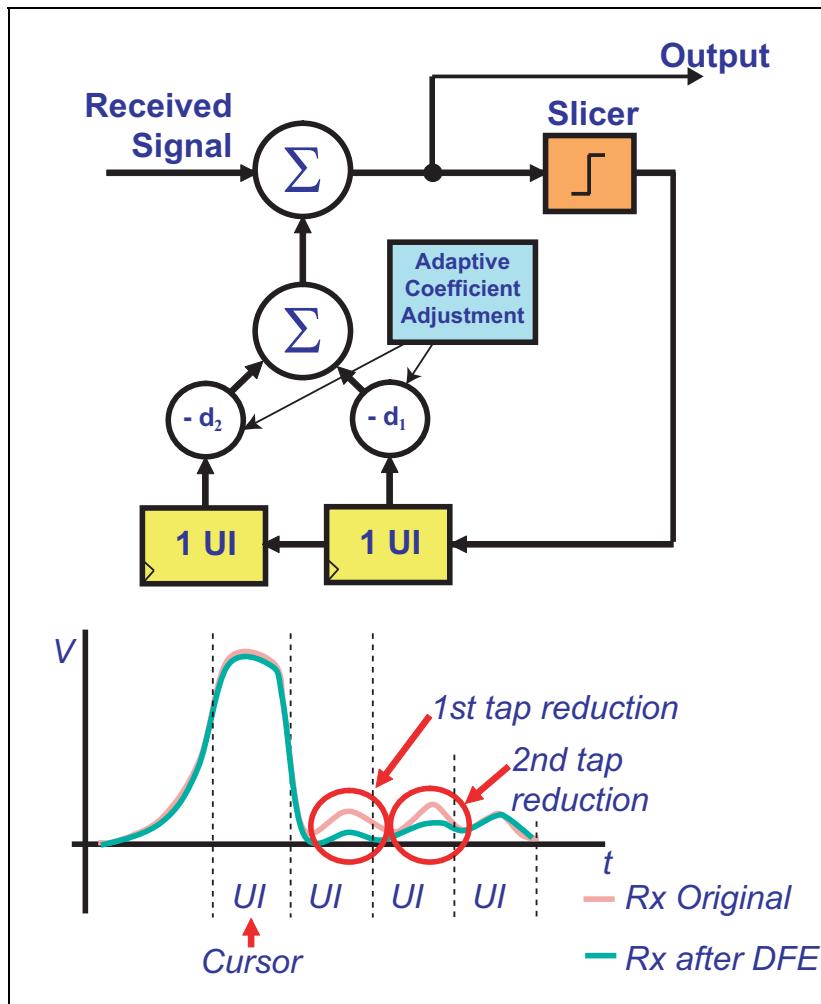
The spec only describes a single-tap filter, but a two-tap version is shown in Figure 13-33 on page 497 to illustrate another option. The motivation for including more taps is to create a cleaner output, since each tap reduces the noise for one more UI. Thus, two taps further reduce the undesirable components of the signal, as shown in the pulse response waveform at the bottom of the drawing. This version is also shown as adaptive, meaning it's able to modify the coefficient values on the fly based on design-specific criteria.

The coefficients of the filter could be fixed, but if they're adjustable the receiver is allowed to change them at any time as long as doing so doesn't interfere with the current operation. In the section called "Recovery Equalization" on page 587, Receiver Preset Hints are described as being delivered by the Downstream Port to the Upstream Port on a Link, using EQ TS1s. The preset gives a hint, in terms of dB reduction, at a starting point for choosing these coefficients.

Since the spec doesn't require it, what the Receiver chooses to do regarding signal compensation will be implementation specific. Industry literature states that DFE is more effective when working with an open eye, and that's why it's usually employed after a linear equalizer that serves to clean up the input enough for DFE to work well.

## Chapter 13: Physical Layer - Electrical

Figure 13-33: Rx 2-Tap DFE



### Receiver Characteristics

Some selected Receiver characteristics are listed in Table 13-5 on page 498. The Receiver Eye Diagram in Figure 13-34 on page 499 also illustrates some of the parameters listed in the table.

# PCI Express Technology

---

*Table 13-5: Common Receiver Characteristics*

Item	2.5 GT/ s.	5.0 GT/s.	8.0 GT/s	Units	Notes
UI	399.88 (min) 400.12 (max)	199.94 (min) 200.06 (max)	124.9625 (min) 125.0375 (max)	ps	Unit Interval = bit time.
T <sub>RX-EYE</sub>	0.4 (min)	Indirectly specified		UI	Minimum eye width for a BER or $10^{-12}$ . At higher rates and long channels the eye is effectively closed, making external mea- surement impractical.
V <sub>RX-EYE</sub>	300	120 (CC) 100 (DC)	Not specified	mV <sub>pp</sub> diff	CC = common clocked, DC = data clocked
V <sub>RX-DIFF- PP-CC</sub>	175 (min) 1200 (max)	120 (min) 1200 (max)	Indirectly specified	mV	Peak-to-peak differential voltage sensitivity of common-clocked Receiver.
V <sub>RX-DIFF- PP-DC</sub>	175 (min) 1200 (max)	100 (min) 1200 (max)	Indirectly specified	mV	Peak-to-peak differential voltage sensitivity of data-clocked Receiver.
V <sub>RX-IDLE- DET-DIFF<sub>P-P</sub></sub>	65 (min) 175 (max)			mV	Electrical Idle detect threshold at the Receiver pins.
Z <sub>RX-DIFF- DC</sub>	80 (min) 120 (max)	Covered by RL <sub>RX-DIFF</sub>		$\Omega$	At higher frequencies im- pedance can no longer be repre- sented by a lumped-sum value and must be described in more detail.
Z <sub>RX--DC</sub>	40 (min) 60 (max)	40 (min) 60 (max)	Bounded by RL <sub>RX-CM</sub>	$\Omega$	DC impedance needed for Receiver Detect.

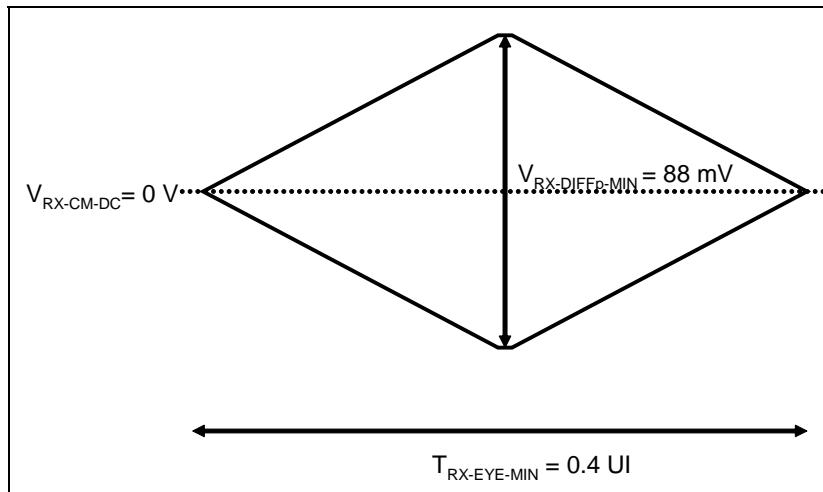
## Chapter 13: Physical Layer - Electrical

---

*Table 13-5: Common Receiver Characteristics (Continued)*

Item	2.5 GT/ s.	5.0 GT/s.	8.0 GT/s	Units	Notes
$L_{RX-SKEW}$	20	8	6	ns	Max Lane-to-Lane skew that a Receiver must be able to correct.
$RL_{RX-DIFF}$	10 (min)	10 (min) for 0.05 - 1.25 GHz, 8 (min) for >1.25 - 2.5 GHz	10 (min) for 0.05 - 1.25 GHz, 8 (min) for >1.25 - 2.5 GHz, 5 (min) for >2.5 - 4.0 GHz	dB	Rx package + Si differential return loss
$RL_{RX-CM}$	6 (min)	6 (min)	6 (min) for 0.05 - 2.5 GHz, 5 (min) for >2.5 - 4 GHz	dB	Common mode Rx return loss

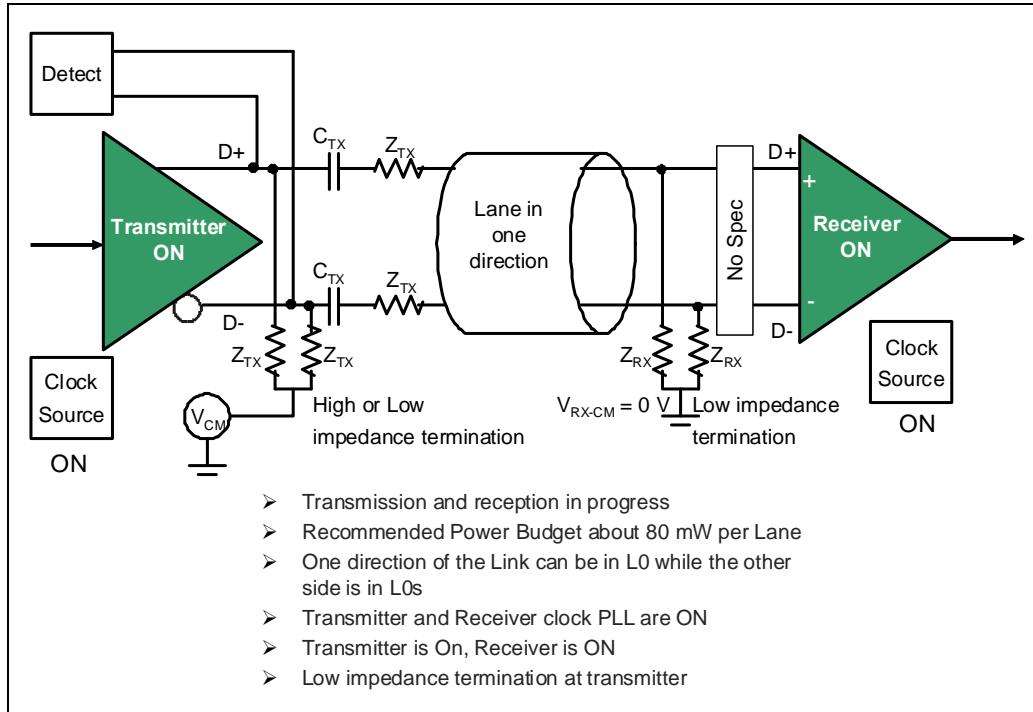
*Figure 13-34: 2.5 GT/s Receiver Eye Diagram*



## Link Power Management States

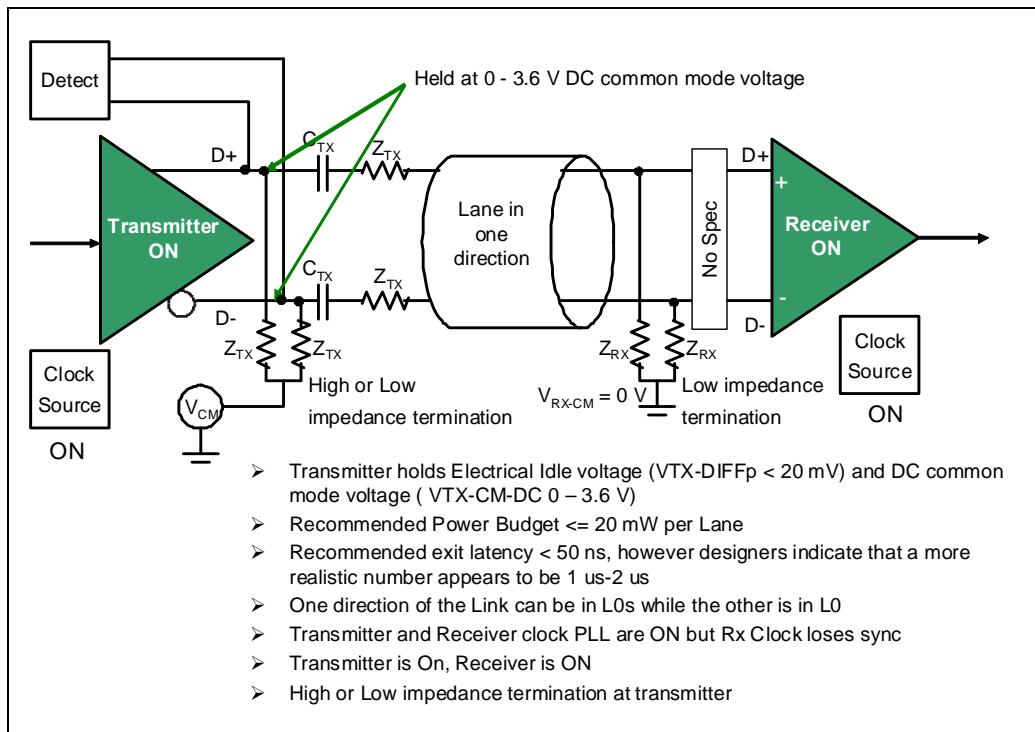
Figure 13-35 on page 500 through Figure 13-39 on page 504 illustrate the electrical state of the Physical Layer while the link is in various power management states and describe several characteristics. One of these is the Tx and Rx terminations, which are sometimes implemented as active logic

Figure 13-35: L0 Full-On Link State



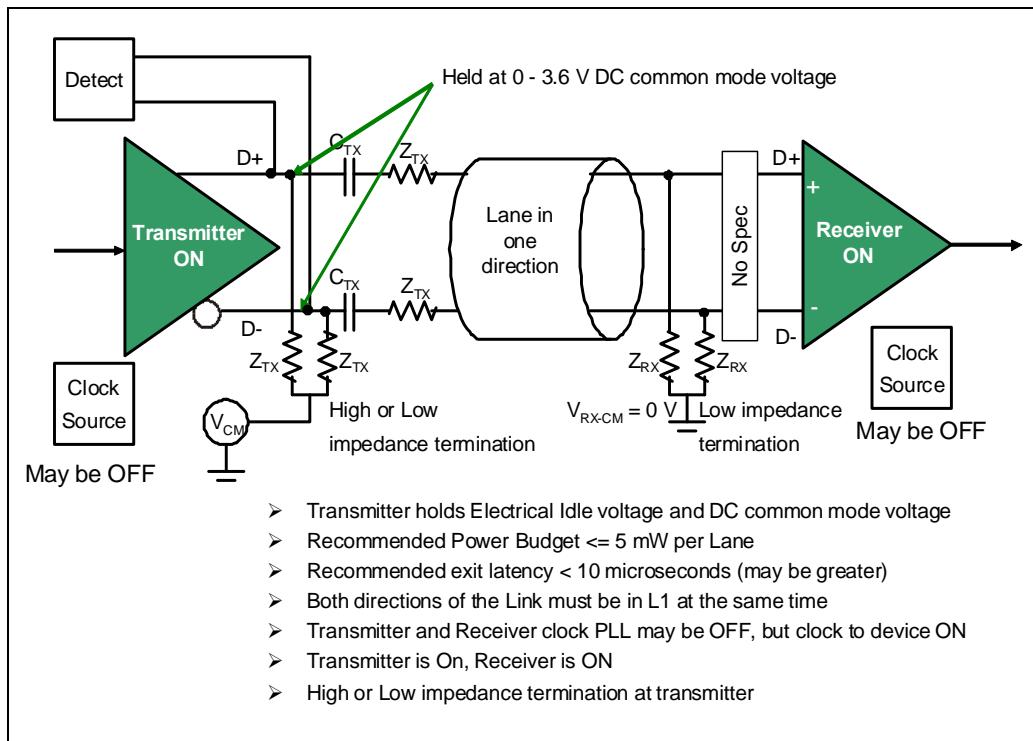
## Chapter 13: Physical Layer - Electrical

Figure 13-36: L0s Low Power Link State



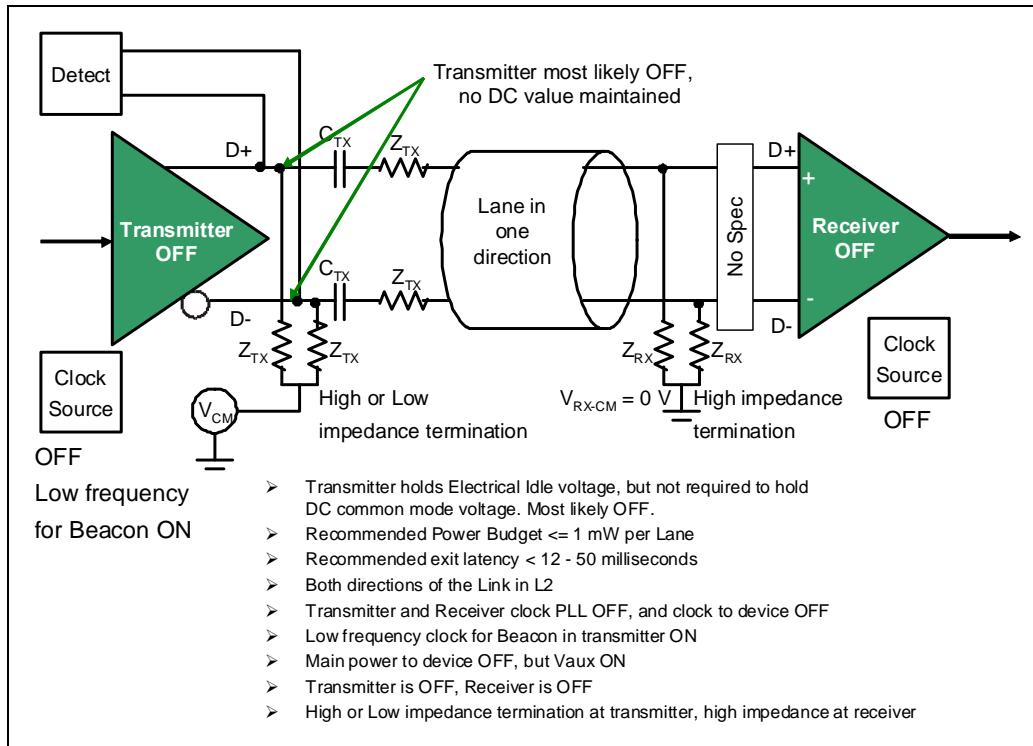
# PCI Express Technology

Figure 13-37: L1 Low Power Link State



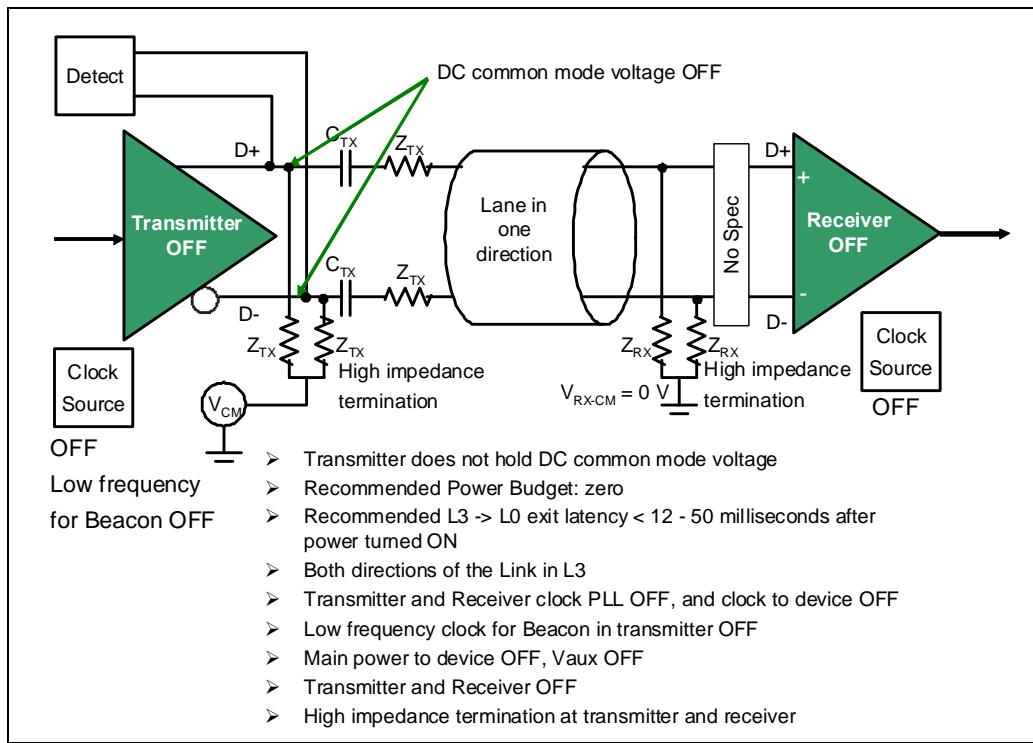
# Chapter 13: Physical Layer - Electrical

Figure 13-38: L2 Low Power Link State



# PCI Express Technology

Figure 13-39: L3 Link Off State



---

# **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**

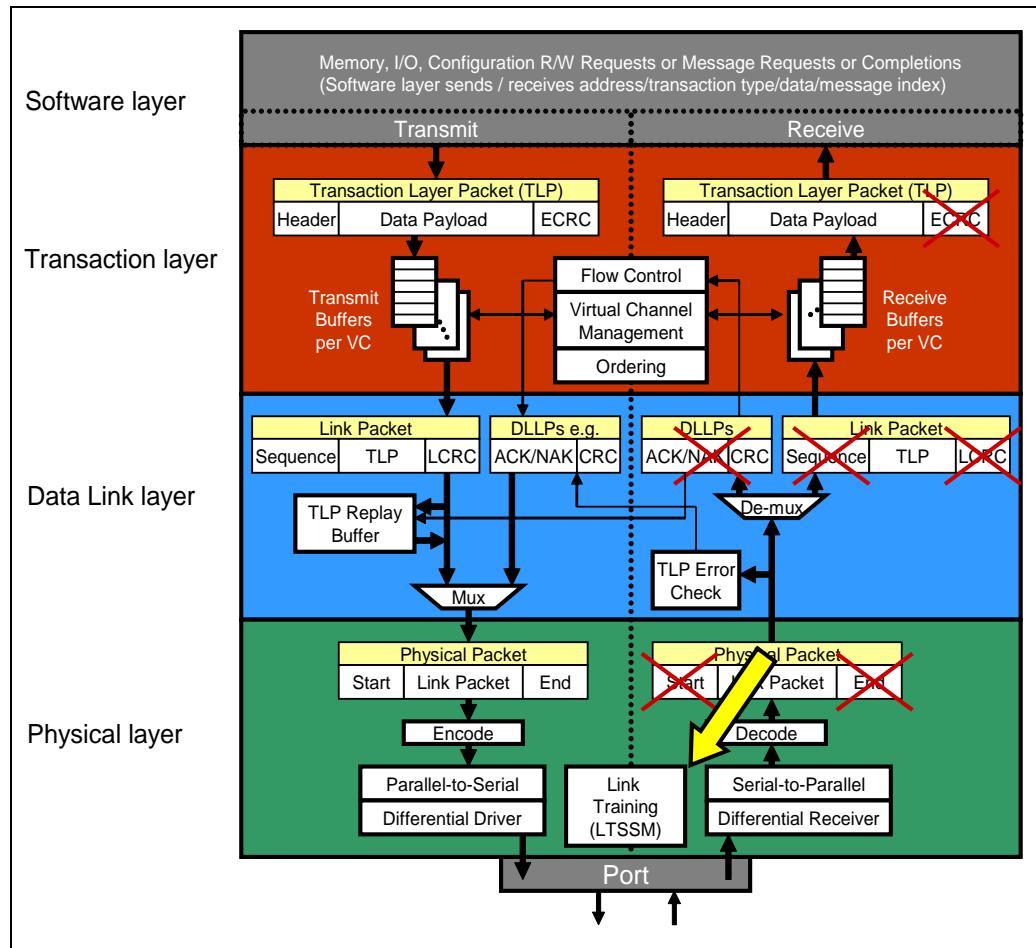
The next chapter discusses error types that occur in a PCIe Port or Link, how they are detected, reported, and options for handling them. Since PCIe is designed to be backward compatible with PCI error reporting, a review of the PCI approach to error handling is included as background information. Then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

# PCI Express Technology

## 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



## Chapter 14: Link Initialization & Training

---

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.

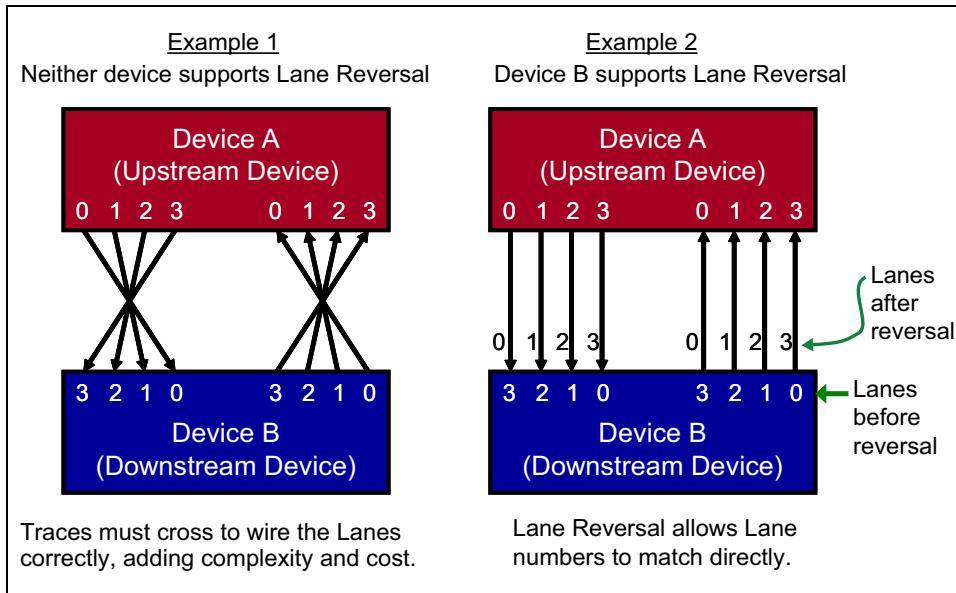
Several things are configured during the Link initialization and training process. Let's consider what they are and define some terms up front.

- **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-

# PCI Express Technology

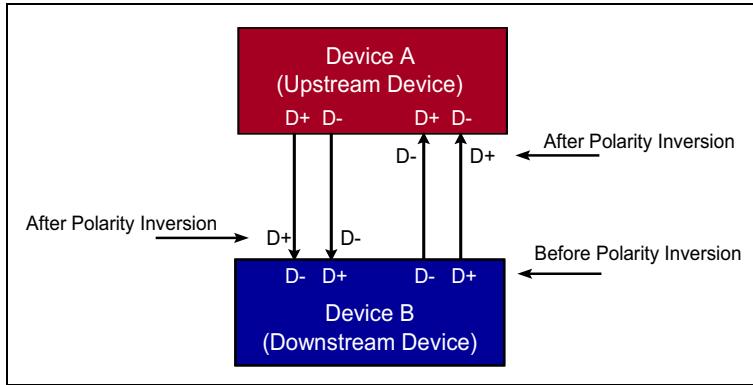
ing Link training and one device must internally reverse its Lane numbering. Since the spec doesn't require support for this, board designers will need to verify that at least one of the connected devices supports this feature before wiring the Lanes in reverse order.

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



# Chapter 14: Link Initialization & Training

Figure 14-3: Polarity Inversion Example (Support Required)



- **Link Data Rate:** After a reset, Link initialization and training will always use the default 2.5Gbit/s data rate for backward compatibility. If higher data rates are available, they are advertised during this process and, when the training is completed, devices will automatically go through a quick re-training to change to the highest commonly supported rate.
- **Lane-to-Lane De-skew:** Trace length variations and other factors cause the parallel bit streams of a multi-Lane Link to arrive at the Receivers at different times, a problem referred to as signal skew. Receivers are required to compensate for this skew by delaying the early arrivals as needed to align the bit streams (see “Lane-to-Lane Skew” on page 442). They must correct a relatively big skew automatically (20ns difference in arrival time is permitted at 2.5GT/s), and that frees board designers from the sometimes difficult constraint of creating equal-length traces. Together with Polarity Inversion and Lane Reversal, this greatly simplifies the board designer’s task of creating a reliable high-speed Link.

---

## Ordered Sets in Link Training

---

### General

All of the different types of Physical Layer Ordered Sets were described in the section called “Ordered sets” on page 388. Training Sequences TS1 and TS2 are of interest during the training process. The format for these when in Gen1 or Gen2 mode is shown in Figure 14-4 on page 510, while for Gen3 mode of operation, they are as shown in Figure 14-5 on page 511. A detailed description of their contents follows.

# PCI Express Technology

---

Figure 14-4: TS1 and TS2 Ordered Sets When In Gen1 or Gen2 Mode

0	COM	K28.5
1	Link #	0 - 255 = D0.0 - D31.7, PAD = K23.7
2	Lane #	0 - 31 = D0.0 - D17.1, PAD = K23.7
3	# FTS	# of FTSs required by Receiver for L0s recovery
4	Rate ID	Bit 1 must be set, indicates 2.5 GT/s support
5	Train Ctl	
6	TS ID or EQ Info	Equalization info when changing to 8.0 GT/s, else TS1 or TS2 Identifier
9		
10	TS ID	TS1 Identifier = D10.2 TS2 Identifier = D5.2
15		

---

## TS1 and TS2 Ordered Sets

As seen in the illustrations, TS1s and TS2s consist of 16 Symbols. They are exchanged during the Polling, Configuration, and Recovery states of the LTSSM described in “Link Training and Status State Machine (LTSSM)” on page 518. The Symbols are described below and summarized in Table 14-1 on page 514 for TS1s and Table 14-2 on page 516 for TS2s.

To make the descriptions a little shorter and easier to read, the term “Gen1” will be used to indicated data rate of 2.5 GT/s, “Gen2” to indicated data rate of 5.0 GT/s and “Gen3” to indicate data rates of 8.0 GT/s. Also, note that the PAD character used in the Link and Lane numbers is represented by the K23.7 character for the lower data rates, but as the data byte F7h for Gen3. In our discussion the distinction between the types of PAD is not interesting and will simply be implied.

## Chapter 14: Link Initialization & Training

Figure 14-5: TS1 and TS2 Ordered Set Block When In Gen3 Mode of Operation

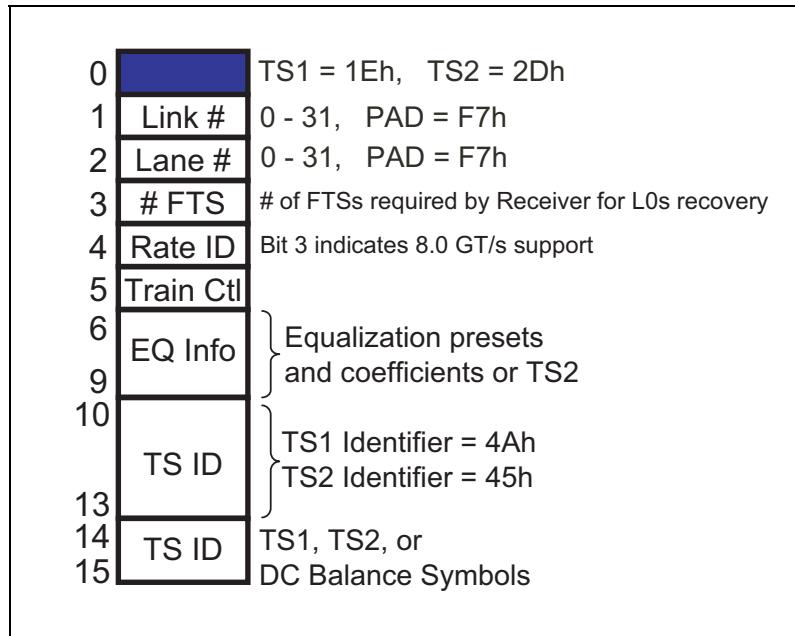


Table 14-1 on page 514 and Table 14-2 on page 516 is a summary of TS1 and TS2 contents. A more detailed description of the 16 TS1/TS2 Symbols follows:

- **Symbol 0:**
  - For **Gen1 or Gen2**, the first Symbol of any Ordered Set is the K28.5 (COM) character. Receivers use this character to acquire Symbol Lock. Since it must appear on all Lanes at the same time it's also useful for de-skewing the Lanes.
  - For **Gen3**, an Ordered Set is identified by the 2-bit Sync Header that must precede the Block (not shown in the illustration), and the first Symbol after that indicates which Ordered Set will follow. For a TS1, the first Symbol is 1Eh, and for a TS2, it's 2Dh.
- **Symbol 1 (Link #):** In the Polling state this field contains the PAD Symbol, but in the other states a Link Number is assigned.
- **Symbol 2 (Lane #):** In the Polling state this field contains the PAD Symbol, but in the other states a Lane Number is assigned.
- **Symbol 3 (N\_FTS):** Indicates the number of Fast Training Sequences the Receiver will need in order to achieve the L0 state when exiting from the L0s power state at the current speed. Transmitters will send at least that many

# PCI Express Technology

---

FTSs to exit L0s. The amount of time needed for this depends on how many are needed and the data rate in use. For example, at 2.5 GT/s each Symbol takes 4ns so, if 200 FTSs were needed the required time would be  $200 \text{ FTS} * 4 \text{ Symbols per FTS} * 4\text{ns/Symbol} = 3200 \text{ ns}$ . If the Extended Synch bit is set in the transmitter device, a total of 4096 FTSs must be sent. This large number is intended to provide enough time for external Link monitoring tools to acquire Bit and Symbol Lock, since some of them may be slow in this regard.

- **Symbol 4 (Rate ID):** Devices report which data rates they support, along with a little more information used for hardware-initiated bandwidth changes. The 2.5 GT/s rate must always be supported and the Link will always train to that speed automatically after reset so that newer components will remain backward compatible with older ones. If 8.0 GT/s is supported, it's also required that 5.0 GT/s must be available. Other information in this Symbol includes the following:
  - **Autonomous Change:** If set, any requested bandwidth change was initiated for power-management reasons. If a change is requested and this bit is not set, then unreliable operation has been detected at the higher speed or wider Link and the change is requested to fix that problem.
  - **Selectable De-emphasis**
    - **Upstream Ports** set this to indicate their desired de-emphasis level at 5.0 GT/s. How they make this choice is implementation specific. In the Recovery.RcvrCfg state, they register the value they receive for this bit internally (the spec describes it as being stored in a select\_deemphasis variable).
    - **Downstream Ports and Root Ports:** In the Polling.Compliance state the select\_deemphasis variable must be set to match the received value of this bit. In the Recovery.RcvrCfg state, the Transmitter sets this bit in its TS2s to match the Selectable De-emphasis field in the Link Control 2 register. Since this register bit is hardware-initialized, the expectation is that it's assigned to an optimal value at power-up by firmware or a strapping option.
    - In Loopback mode at 5.0 GT/s, the Slave de-emphasis value is assigned by this bit in the TS1s sent by the Master.
  - **Link Upconfigure Capability:** Reports whether a wide Link whose width is reduced will be capable of going back to the wide case or not. If both sides of a Link report this during Configuration.Complete, this fact is recorded internally (e.g. an upconfigure\_capable bit is set).
- **Symbol 5 (Training Control):** Communicates special conditions such as a Hot Reset, Enable Loopback mode, Disable Link, Disable Scrambling.

# Chapter 14: Link Initialization & Training

---

- **Symbols 6-9 (Equalization Control):**
  - For **Gen1 or Gen2**, Symbols 7-9 are just TS1 or TS2 indicators, and Symbol 6 usually is, too. However, if bit 7 of Symbol 6 is set to one instead of the zero that would be there for the TS1 or TS2 identifier, that indicates that this is an EQ TS1 or EQ TS2 sent from the Downstream Port (DSP - port that faces downstream, like a Root Port). The “EQ” label stands for equalization, and means that the Link is going to change to 8.0 GT/s and so the Upstream Port (USP - port that faces upstream, like an Endpoint Port) needs to know what equalizer values to use. For EQ TS1s or TS2s, Symbol 6 gives that information to the USP in the form of Transmitter Presets and Receiver Preset Hints. Ports that support 8.0 GT/s must accept either TS type (regular or EQ), but ports that do not support it are not required to accept the EQ type. The possible values for these presets are listed in Table 14-8 on page 579 and Table 14-9 on page 580.
  - For **Gen3**, Symbols 6-9 provide Preset values and Coefficients for the Equalization process. Bit 7 of Symbol 6 in a TS2 can now be used by a USP to request that equalization be redone. If it does, bit 6 may also be set to indicate that the time needed to repeat the equalization process won’t cause problems, such as a completion timeout, as long as it’s done quickly (within 1ms of returning to L0). This might be needed, for example, if a problem was detected with the equalization results. A DSP can also use bits 6 and 7 to ask the USP to make such a request and guarantee no side effects, although the USP is not required to respond to this. For more on the equalization process, see “Link Equalization Overview” on page 577.
- **Symbols 10-13:** TS1 or TS2 identifiers.
- **Symbols 14-15:** (DC Balance)
  - For **Gen1 and Gen2**, these are just TS1 or TS2 indicators since DC Balance is maintained by 8b/10b encoding.
  - For **Gen3**, the contents of these two Symbols depend on the DC Balance of the Lane. Each Lane of a Transmitter must independently track the running DC Balance for all the scrambled bits sent for TS1s and TS2s. “Running DC Balance” means the difference between the number of ones sent vs. the number of zeroes sent, and Lanes must be capable of tracking a difference of up to 511 in either direction. These counters saturate at their max value but continue to track reductions. For example, if the counter indicates that 511 more ones than zeroes have been sent, then no matter how many more ones are sent, the value will stay at 511. However, if 2 zeroes are sent, the counter will count down to 509. When a TS1 or TS2 is sent, the following algorithm is used to determine Symbols 14 and 15:
    - If the running DC Balance value is > 31 at the end of Symbol 11 and more ones have been sent, Symbol 14 = 20h and Symbol 15 = 08h. If more zeroes have been sent, Symbol 14 = DFh and Symbol 15 = F7h.

# PCI Express Technology

---

- If the running DC Balance value is > 15, Symbol 14 = the normal scrambled TS1 or TS2 identifier, while Symbol 15 = 08h to reduce the number of ones, or F7h to reduce the number of zeroes in the DC Balance count.
- Otherwise, the normal TS1 or TS2 identifier Symbols will be sent.
- Other notes on Gen3 DC Balance:
  - The running DC Balance is reset by an exit from Electrical Idle or an EIEOS after a Data Block.
  - The DC Balance Symbols bypass scrambling to ensure that the expected bit pattern is sent.

*Table 14-1: Summary of TS1 Ordered Set Contents*

Symbol Number	Description
0	<ul style="list-style-type: none"><li>• For Gen1 or Gen2, the COM (K28.5) Symbol</li><li>• For Gen3, 1Eh indicates a TS1.</li></ul>
1	Link Number <ul style="list-style-type: none"><li>• Ports that don't support Gen3: 0-255, PAD</li><li>• Downstream ports that support Gen3: 0-31, PAD</li><li>• Upstream ports that support Gen3: 0-255, PAD</li></ul>
2	Lane Number <ul style="list-style-type: none"><li>• 0-31, PAD</li></ul>
3	N_FTS <ul style="list-style-type: none"><li>• Number of FTS Ordered Sets required by receiver to achieve L0 when exiting L0s: 0 - 255</li></ul>
4	Data Rate Identifier: <ul style="list-style-type: none"><li>• Bit 0 — Reserved.</li><li>• Bit 1 — 2.5 GT/s supported (must be set to 1b)</li><li>• Bit 2 — 5.0 GT/s supported (must be set if bit 3 is set)</li><li>• Bit 3 — 8.0 GT/s supported</li><li>• Bits 5:4 — Reserved</li><li>• Bit 6 — Autonomous Change&gt;Selectable De-emphasis<ul style="list-style-type: none"><li>– Downstream Ports: Used in Polling.Active, Configuration.Linkwidth.Start, and Loopback.Entry LTSSM states, and reserved in all other states.</li><li>– Upstream Ports: Used in Polling.Active, Configuration, Recovery, and Loopback.Entry LTSSM states and reserved in all other states.</li></ul></li><li>• Bit 7 — Speed change. This can only be set to one in the Recovery.RcvrLock LTSSM state, and is reserved in all other states.</li></ul>

## Chapter 14: Link Initialization & Training

---

*Table 14-1: Summary of TS1 Ordered Set Contents (Continued)*

Symbol Number	Description
5	<p>Training Control (0=De-assert, 1 = Assert)</p> <ul style="list-style-type: none"> <li>• Bit 0 – Hot Reset</li> <li>• Bit 1 – Disable Link</li> <li>• Bit 2 – Loopback</li> <li>• Bit 3 – Disable Scrambling (for 2.5 or 5.0 GT/s; reserved for Gen3)</li> <li>• Bit 4 – Compliance Receive (optional for 2.5 GT/s, required for all other rates)</li> <li>• Bits 7:5 – Reserved, Set to 0</li> </ul>
6	<p>For Gen1 or Gen2:</p> <ul style="list-style-type: none"> <li>• TS1 identifier (4Ah) encoded as D10.2</li> <li>• EQ TS1s encode this as</li> <li>    Bits 2:0 – Receiver preset hint</li> <li>    Bits 6:3 – Transmitter Preset</li> <li>    Bit 7 – set to 1b</li> </ul> <p>For Gen3:</p> <ul style="list-style-type: none"> <li>• Bits 1:0 – Equalization Control (EC). Only used in Recovery.Equalization and Loopback LTSSM states; must be 00b in all other states.</li> <li>• Bit 2 – Reset EIEOS Interval Count. Only used in Recovery.Equalization LTSSM state; reserved in all other states.</li> <li>• Bits 6:3 – Transmitter Preset</li> <li>• Bit 7 – Use Preset. (If one, use the preset values instead of the coefficient values. If zero, use the coefficients rather than the presets.) Only used in Recovery.Equalization and Loopback LTSSM states; reserved in all other states.</li> </ul>
7	<p>For Gen1 or Gen2 GT/s, TS1 identifier (4Ah) encoded as D10.2</p> <p>For Gen3:</p> <ul style="list-style-type: none"> <li>• Bits 5:0 – FS (Full Swing value) when the EC field of Symbol 6 is 01b, otherwise, Pre-cursor Coefficient.</li> <li>• Bits 7:6 – Reserved.</li> </ul>
8	<p>For Gen1 or Gen2, TS1 identifier (4Ah) encoded as D10.2</p> <p>For Gen3:</p> <ul style="list-style-type: none"> <li>• Bits 5:0 – LF (Low Frequency value) when the EC field of Symbol 6 is 01b, otherwise, Cursor Coefficient.</li> <li>• Bits 7:6 – Reserved.</li> </ul>

# PCI Express Technology

---

Table 14-1: Summary of TS1 Ordered Set Contents (Continued)

Symbol Number	Description
9	For Gen1 or Gen2, TS1 identifier (4Ah) encoded as D10.2 For Gen3: <ul style="list-style-type: none"><li>• Bits 5:0 — Post-cursor Coefficient.</li><li>• Bit 6 — Reject Coefficient Values. Only set in specific Phases of the Recovery Equalization LTSSM state; must be 0b otherwise.</li><li>• Bit 7 — Parity (P) This is the even parity of all bits of Symbols 6, 7, and 8 and bits 6:0 of Symbol 9. Receivers must calculate this and compare it to the received Parity bit. Received TS1s are only valid if the Parity bits match.</li></ul>
10-13	For Gen1 or Gen2, TS1 identifier (4Ah) encoded as D10.2 <ul style="list-style-type: none"><li>• For Gen3, TS1 identifier (4Ah)</li></ul>
14-15	For Gen1 or Gen2, TS1 identifier (4Ah) encoded as D10.2 <ul style="list-style-type: none"><li>• For Gen3, TS1 identifier (4Ah), or a DC-Balance Symbol.</li></ul>

The observant reader may wonder why EQ TS1s are shown in Symbol 6 for the lower data rates since only 8.0 GT/s data rates use equalization. That's because they're used to deliver EQ values for Lanes that support Gen3 but are currently operating at a lower rate and want to change to 8.0 GT/s. For more details regarding this and the Equalization process for Gen3 in general, see "Link Equalization Overview" on page 577.

Table 14-2: Summary of TS2 Ordered Set Contents

Symbol Number	Description
0	<ul style="list-style-type: none"><li>• For Gen1 or Gen2, the COM (K28.5) Symbol</li><li>• For Gen3, 2Dh indicates a TS2.</li></ul>
1	Link Number <ul style="list-style-type: none"><li>• Ports that don't support Gen3: 0-255, PAD</li><li>• Downstream ports that support Gen3: 0-31, PAD</li><li>• Upstream ports that support Gen3 0-255, PAD</li></ul>
2	Lane Number <ul style="list-style-type: none"><li>• 0-31, PAD</li></ul>

## Chapter 14: Link Initialization & Training

---

*Table 14-2: Summary of TS2 Ordered Set Contents (Continued)*

Symbol Number	Description
3	N_FTS <ul style="list-style-type: none"> <li>• Number of FTS Ordered Sets required by receiver to achieve L0 when exiting L0s: 0 - 255</li> </ul>
4	Data Rate Identifier: <ul style="list-style-type: none"> <li>• Bit 0 — Reserved.</li> <li>• Bit 1 — 2.5 GT/s supported (must be set to 1b)</li> <li>• Bit 2 — 5.0 GT/s supported (must be set if bit 3 is set)</li> <li>• Bit 3 — 8.0 GT/s supported</li> <li>• Bits 5:4 — Reserved</li> <li>• Bit 6 — Autonomous Change&gt;Selectable De-emphasis/Link Upconfigure Capability. Used in Polling.Configuration, Configuration.Complete, and Recovery LTSSM states; reserved in all other states.</li> <li>• Bit 7 — Speed change. This can only be set to one in the Recovery.RcvrLock LTSSM state, and is reserved in all other states.</li> </ul>
5	Training Control (0 = De-assert, 1 = Assert) <ul style="list-style-type: none"> <li>• Bit 0 — Hot Reset,</li> <li>• Bit 1 — Disable Link</li> <li>• Bit 2 — Loopback</li> <li>• Bit 3 — Disable Scrambling (for 2.5 or 5.0 GT/s; reserved for Gen3)</li> <li>• Bits 7:4 — Reserved, Set to 0</li> </ul>
6	For Gen1 or Gen2: <ul style="list-style-type: none"> <li>• TS2 identifier (4Ah) encoded as D10.2</li> <li>• EQ TS2s encode this as <ul style="list-style-type: none"> <li>Bits 2:0 — Receiver preset Hint</li> <li>Bits 6:3 — Transmitter Preset</li> <li>Bit 7 — Equalization Command</li> </ul> </li> </ul> For Gen3: <ul style="list-style-type: none"> <li>• Bits 5:0 — Reserved.</li> <li>• Bit 6 — Quiesce Guarantee. Defined for use in Recovery.RcvrCfg only; reserved in all other states.</li> <li>• Bit 7 — Request Equalization. Defined for use in Recovery.RcvrCfg only; reserved in all other states.</li> </ul>
7-13	<ul style="list-style-type: none"> <li>• For Gen1 or Gen2, TS2 identifier (45h) encoded as D5.2</li> <li>• For Gen3, TS2 identifier (45h)</li> </ul>
14-15	<ul style="list-style-type: none"> <li>• For Gen1 or Gen2, TS2 identifier (45h) encoded as D5.2</li> <li>• For Gen3, TS2 identifier (45h), or a DC-Balance Symbol</li> </ul>

## Link Training and Status State Machine (LTSSM)

---

### General

Figure 14-6 on page 519 illustrates the top-level states of the Link Training and Status State Machine (LTSSM). Each state consists of substates. The first LTSSM state entered after exiting Fundamental Reset (Cold or Warm Reset) or Hot Reset is the Detect state.

The LTSSM consists of 11 top-level states: Detect, Polling, Configuration, Recovery, L0, L0s, L1, L2, Hot Reset, Loopback, and Disable. These can be grouped into five categories:

1. Link Training states
2. Re-Training (Recovery) state
3. Software driven Power Management states
4. Active-State Power Management (ASPM) states
5. Other states

When exiting from any type of Reset, the flow of the LTSSM follows the **Link Training states**: Detect => Polling => Configuration => L0. In L0 state, normal packet transmission/reception is in progress.

The **Link Re-Training also called Recovery** state is entered for a variety of reasons, such as changing back from a low-power Link state, like L1, or changing the Link bandwidth (through speed or width changes). In this state, the Link repeats as much of the training process as needed to handle the matter and returns to L0 (normal operation).

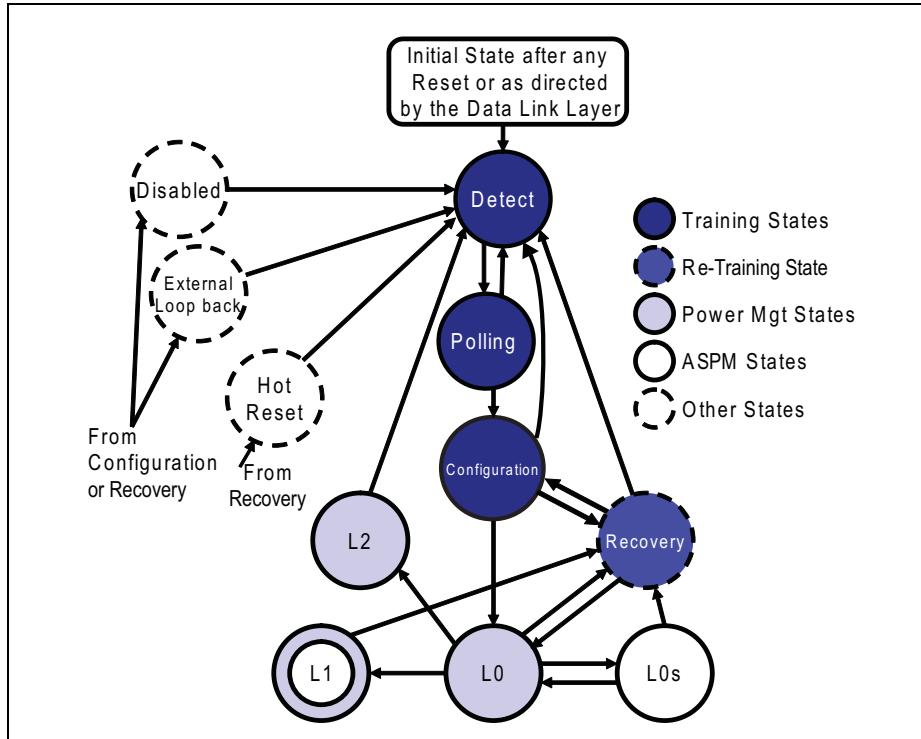
Power management software may also place a device into a low-power device state (D1, D2, D3<sub>Hot</sub> or D3<sub>Cold</sub>) and that will force the Link into a lower **Power Management Link state** (L1 or L2).

If there are no packets to send for a time, ASPM hardware may be allowed to automatically transition the Link into low power **ASPM states** (L0s or ASPM L1).

In addition, software can direct a Link to enter some **other special states**: Disabled, Loopback, or Hot Reset. Here, these are collectively called the Other states group.

# Chapter 14: Link Initialization & Training

Figure 14-6: Link Training and Status State Machine (LTSSM)



## Overview of LTSSM States

Below is a brief description of the 11 high-level LTSSM states.

- **Detect:** The initial state after reset. In this state, a device electrically detects a Receiver is present at the far end of the Link. That's an unusual thing in the world of serial transports, but it's done to facilitate testing, as we'll see in the next state. Detect may also be entered from a number of other LTSSM states as described later.
- **Polling:** In this state, Transmitters begin to send TS1s and TS2s (at 2.5 GT/s for backward compatibility) so that Receivers can use them to accomplish the following:
  - Achieve Bit Lock
  - Acquire Symbol Lock or Block Lock
  - Correct Lane polarity inversion, if needed
  - Learn available Lane data rates

# PCI Express Technology

---

- If directed, Initiate the Compliance test sequence: The way this works is that if a receiver was detected in the Detect state but no incoming signal is seen, it's understood to mean that the device has been connected to a test load. In that case, it should send the specified Compliance test pattern to facilitate testing. This allows test equipment to quickly verify that voltage, BER, timing, and other parameters are within tolerance.
- **Configuration:** Upstream and Downstream components now play specific roles as they continue to exchange TS1s and TS2s at 2.5 GT/s to accomplish the following:
  - Determine Link width
  - Assign Lane numbers
  - Optionally check for Lane reversal and correct it
  - Deskew Lane-to-Lane timing differences

From this state, scrambling can be disabled, the Disable and Loopback states can be entered, and the number of FTS Ordered Sets required to transition from the L0s state to the L0 state is recorded from the TS1s and TS2s.
- **L0:** This is the normal, fully-active state of a Link during which TLPs, DLLPs and Ordered Sets can be exchanged. In this state, the Link could be running at higher speeds than 2.5 GT/s, but only after re-training (Recovery) the Link and going through a speed change procedure.
- **Recovery:** This state is entered when the Link needs re-training. This could be caused by errors in L0, or recovery from L1 back to L0, or recovery from L0s if the Link does not train properly using the FTS sequence. In Recovery, Bit Lock and Symbol/Block Lock are re-established in a manner similar to that used in the Polling state but it typically takes much less time.
- **L0s:** This ASPM state is designed to provide some power savings while affording a quick recovery time back to L0. It's entered when one Transmitter sends the EIOS while in the L0 state. Exit from L0s involves sending FTSs to quickly re-acquire Bit and Symbol/Block Lock.
- **L1:** This state provides greater power savings by trading off a longer recovery time than L0s does (see "Active State Power Management (ASPM)" on page 735). Entry into L1 involves a negotiation between both Link partners to enter it together and can occur in one of two ways:
  - The first is autonomous with ASPM: hardware in an Upstream Port with no scheduled TLPs or DLLPs to transmit can automatically negotiate to put its Link into the L1 state. If the Downstream Port agrees, the Link enters L1. If not, the Upstream Port will enter L0s instead (if enabled).
  - The second is the result of power management software issuing a commanding a device to a low-power state (D1, D2, or D3<sub>Hot</sub>). As a result, the Upstream Port notifies the Downstream Port that they must enter L1, the Downstream Port acknowledges that, and they enter L1.

# Chapter 14: Link Initialization & Training

---

- **L2:** In this state the main power to the devices is turned off to achieve a greater power savings. Almost all of the logic is off, but a small amount of power is still available from the  $V_{aux}$  source to allow the device to indicate a wakeup event. An Upstream Port that supports this wakeup capability can send a very low frequency signal called the Beacon and a Downstream Port can forward it to the Root Complex to get system attention (see “Beacon Signaling” on page 483). Using the Beacon, or a side-band WAKE# signal, a device can trigger a system wakeup event to get main power restored. [An L3 Link power state is also defined, but it doesn’t relate to the LTSSM states. The L3 state is the full-off condition in which  $V_{aux}$  power is not available and a wakeup event can’t be signaled.]
- **Loopback:** This state is used for testing but exactly what a Receiver does in this mode (for example: how much of the logic participates) is left unspecified. The basic operation is simple enough: the device that will be the Loopback Master sends TS1 Ordered Sets that have the Loopback bit set in the Training Control field to the device that will be the Loopback Slave. When a device sees two consecutive TS1s with the Loopback bit set, it enters the Loopback state as the Loopback Slave and echoes back everything that comes in. The Master, recognizing that what it is sending is now being echoed, sends any pattern of Symbols that follow the 8b/10b encoding rules, and the Slave echoes them back exactly as they were sent, providing a round-trip verification of Link integrity.
- **Disable:** This state allows a configured Link to be disabled. In this state, the Transmitter is in the Electrical Idle state while the Receiver is in the low impedance state. This might be necessary because the Link has become unreliable or due to a surprise removal of the device. Software commands a device to do this by setting the Disable bit in the Link Control register. The device then sends 16 TS1s with the Disable Link bit set in the TS1 Training Control field. Receivers are disabled when they receive those TS1s.
- **Hot Reset:** Software can reset a Link by setting the Secondary Bus Reset bit in the Bridge Control register. That causes the bridge’s Downstream Port to send TS1s with the Hot Reset bit set in the TS1 Training Control field (see “Hot Reset (In-band Reset)” on page 837) When a Receiver sees two consecutive TS1s with the Hot Reset bit set, it must reset its device.

---

## Introductions, Examples and State/Substates

The balance of this chapter covers each of the LTSSM states. Depending on the complexity of a given state, the discussion may include an introduction, general background, and/or examples that accompanies the detailed discussion of the State/Substate. In some cases, the reader may choose to skip the detailed cover-

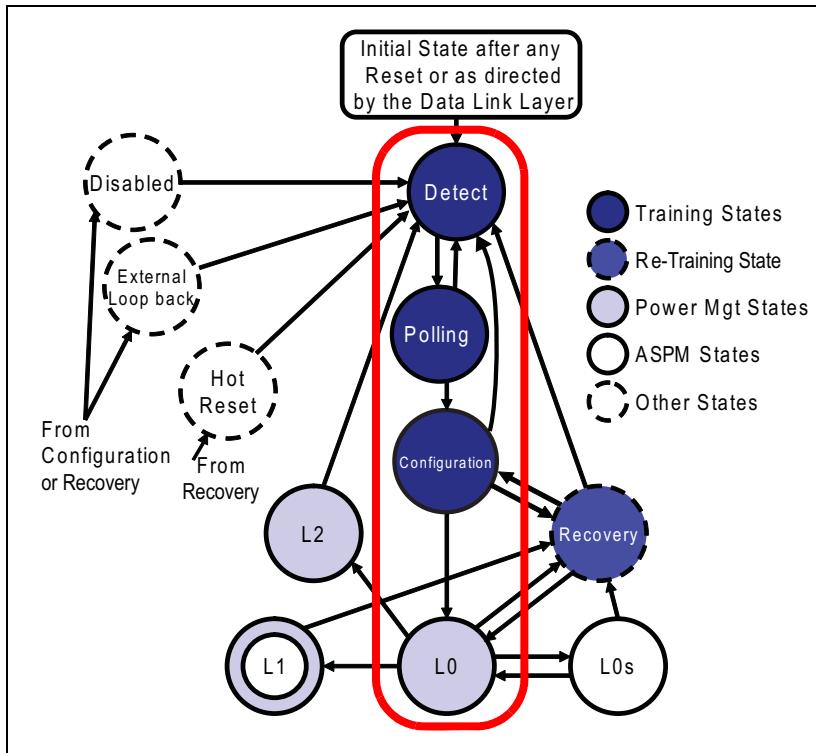
# PCI Express Technology

---

age and jump to introductory material. Each section is organized to facilitate these options.

Every device must perform initial link training at the base rate of 2.5 GT/s. Figure 14-7 highlights the states involved in the initial training sequence. Devices capable of operating at 5.0 or 8.0 GT/s must transition to the Recovery state to change the speed to the higher rate chosen.

Figure 14-7: States Involved in Initial Link Training at 2.5 Gb/s



---

## Detect State

---

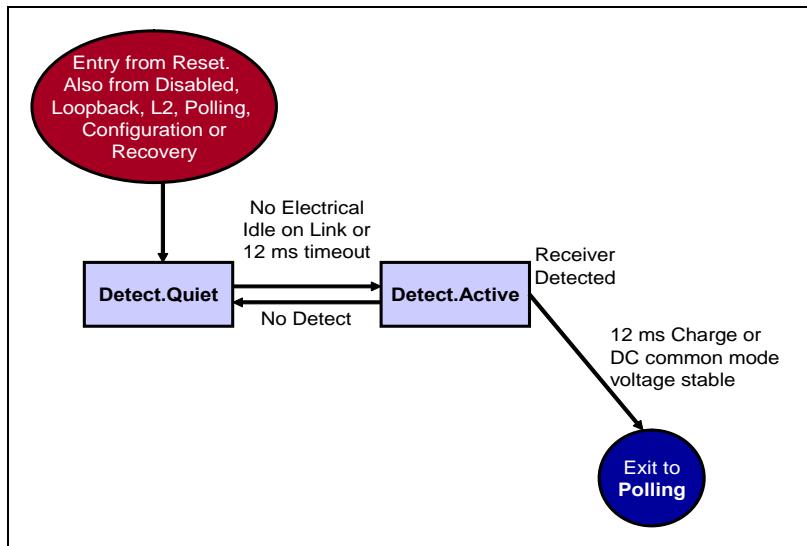
### Introduction

Figure 14-8 represents the two substates and transitions associated with the Detect state. The actions associated with the Detect state are performed by each

# Chapter 14: Link Initialization & Training

transmitter in the process of detecting the presence of a receiver at the opposite end of the link. Because there are only two substates and because they are fairly simple, we will move directly to the substate discussions.

Figure 14-8: Detect State Machine



## Detailed Detect Substate

### Detect.Quiet

This substate is the initial state after any reset (except Function Level Reset) or power-up event and must be entered within 20 ms after Reset. This substate is also entered from other states if unable to move forward (See the states that may enter Detect.Quiet in Figure 14-8 on page 523). The properties of this substate are listed below:

- The Transmitter starts in Electrical Idle (but the DC common mode voltage doesn't have to be within the normally-specified range).
- The intended data rate is set to 2.5 GT/s (Gen1). If it set to a different rate when this substate was entered, the LTSSM must stay in this substate for 1ms before changing the rate to Gen1.
- The Physical Layer's status bit (LinkUp = 0) informs the Data Link Layer that the Link is not operational. The LinkUp status bit is an internal state bit

# PCI Express Technology

---

(not found in standard config space) and also indicates when the Physical Layer has completed Link Training (LinkUp=1), thereby informing the Data Link Layer and Flow Control initialization to begin its part of Link initialization (for more on this, see “The FC Initialization Sequence” on page 223).

- Any previous equalization (Eq.) status is cleared by setting the four Link Status 2 register bits to zero: Eq. Phase 1 Successful, Eq. Phase 2 Successful, Eq. Phase 3 Successful, Eq. Complete.
- Variables:
  - Several variables are cleared to zero: (directed\_speed\_change=0b, upconfigure\_capable=0b, equalization\_done\_8GT\_data\_rate=0b, idle\_to\_rlock\_transitioned=00h). The select\_deemphasis variable setting depends on the port type: for an Upstream Port it’s selected by hardware, while for a Downstream Port it takes the value in the Link Control 2 register of the Selectable Preset/De-emphasis field.
  - Since these variables were defined beginning with the 2.0 spec version, devices designed to earlier spec versions won’t have them and will behave as if directed\_speed\_change and upconfigure\_capable were set to 0b and idle\_to\_rlock\_transitioned was set to FFh.

## *Exit to “Detect.Active”*

The next substate is Detect.Active after a 12 ms timeout or when any Lane exits Electrical Idle.

## **Detect.Active**

This substate is entered from Detect.Quiet. At this time the Transmitter tests whether a Receiver is connected on each Lane by setting a DC common mode voltage of any value in the legal range and then changing it. The detection logic observes the rate of change as the time it takes the line voltage to charge up and compares it to an expected time, such as how long it would take without a Receiver termination. If a Receiver is attached, the charge time will be much longer, making it easy to recognize. For more details on this process, see “Receiver Detection” on page 460. To simplify the discussions that follow, Lanes that detect a Receiver during this substate are referred to as “Detected Lanes.”

## *Exit to “Detect.Quiet”*

If no Lanes detect a Receiver, go back to Detect.Quiet. The loop between them is repeated every 12ms, as long as no Receiver is detected.

## *Exit to “Polling State”*

If a receiver is detected on all Lanes, the next state will be Polling. The Lanes must now drive a DC common voltage within the 0 - 3.6 V  $V_{TX-CM-DC}$  spec.

## *Special Case:*

If some but not all Lanes of a device are connected to a Receiver (like a  $\times 4$

# Chapter 14: Link Initialization & Training

device connected to a x2 device), then wait 12 ms and try it again. If the same Lanes detect a Receiver the second time, exit to the Polling state, otherwise go back to Detect.Quiet. If going to Polling, there are two possibilities for the Lanes that didn't see a Receiver:

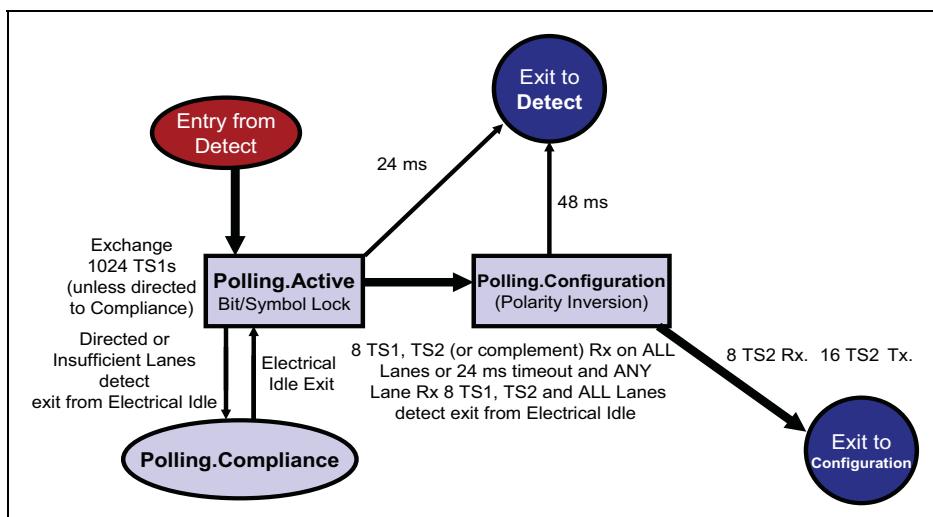
1. If the Lanes can operate as a separate Link (see “Designing Devices with Links that can be Merged” on page 541), use another LTSSM and have those Lanes repeat the detect sequence.
2. If another LTSSM is not available, then the Lanes that don't detect a Receiver will not be part of a Link and must transition to Electrical Idle.

## Polling State

### Introduction

To this point the link has been in the electrical idle state, however during Polling the LTSSM TS1s and TS2s are exchanged between the two connected devices. The primary purpose of this state is for the two devices to understand what the each other is saying. In other words, they need to establish bit and symbol lock on each other's transmitted bit stream and resolve any polarity inversion issues. Once this has been accomplished, each device is successfully receiving the TS1 and TS2 ordered-sets from their link partner. Figure 14-9 on page 525 shows the substates of the Polling state machine.

Figure 14-9: Polling State Machine



## Detailed Polling Substates

### Polling.Active

#### *During Polling.Active*

Transmitters send a minimum of 1024 consecutive TS1s on all detected Lanes once their common-mode voltage has settled at the level specified in the Transmit Margin field. The two Link partners may exit the Detect state at different times, so the TS1 exchange is not synchronized. The time needed to send 1024 TS1s at Gen1 speed (2.5 GT/s) is 64 $\mu$ s.

Some notes regarding this substate are:

- The PAD Symbol must be used in the Lane and Link Number fields of the TS1s.
- All data rates a device supports must be advertised, even if it doesn't intend to use them all.
- Receivers use the incoming TS1s to acquire Bit Lock (see "Achieving Bit Lock" on page 395) and then either Symbol Lock (see "Achieving Symbol Lock" on page 396) for the lower rates, or Block Alignment for 8.0 GT/s (see "Achieving Block Alignment" on page 438).

#### *Exit to "Polling.Configuration"*

The next state is Polling.Configuration if, after sending at least 1024 TS1s **ALL** detected Lanes receive 8 consecutive training sequences (or their complement, due to polarity inversion) that satisfy one of the following conditions:

- TS1s with Link and Lane set to PAD were received with the Compliance Receive bit cleared to 0b (bit 4 of Symbol 5).
- TS1s with Link and Lane set to PAD were received with the Loopback bit of Symbol 5 set to 1b.
- TS2s were received with Link and Lane set to PAD.

If the conditions above are not met, then after a 24ms timeout, if at least 1024 TS1s were sent after receiving a TS1, and **ANY** detected Lane received eight consecutive TS1 or TS2 Ordered Sets (or their complement) with the Lane and Link numbers set to PAD, and one of the following is true:

- TS1s with Link and Lane set to PAD were received with the Compliance Receive (bit 4 of Symbol 5) cleared to 0b.
- TS1s with Link and Lane set to PAD were received with the Loopback (bit 2 of Symbol 5) set to 1b.
- TS2s were received with Link and Lane set to PAD.

## **Chapter 14: Link Initialization & Training**

---

If still none of the conditions above are met, if at least a predetermined number of detected Lanes also detected an exit from Electrical Idle at least once since entering Polling.Active (this prevents one or more bad Transmitters or Receivers from holding up Link configuration). The exact set of predetermined Lanes is implementation specific now, which is a change from the 1.1 spec that needed to see an Electrical Idle exit on all detected Lanes.

### *Exit to “Polling.Compliance”*

If the Enter Compliance bit in the Link Control 2 register is set to 1b, or if this bit was set before entering Polling.Active, the change to Polling.Compliance must be immediate and no TS1s are sent in Polling.Active.

Otherwise, after a 24ms timeout, if:

- All Lanes from the predetermined set have not seen an exit from Electrical Idle since entering Polling.Active (indicates a passive test load such as a resistor on at least one Lane forces all Lanes into Polling.Compliance).
- Any detected Lane received 8 consecutive TS1s (or their complement) with Link and Lane numbers set to PAD, the Compliance Receive bit of Symbol 5 set to 1b and the Loopback bit cleared to 0b.

### *Exit to “Detect State”*

If, after 24ms, the conditions for going to Polling.Configuration or Polling.Compliance are not met, return to the Detect state.

## **Polling.Configuration**

In this substate, a transmitter will stop sending TS1s and start sending TS2s, still with PAD set for the Link and Lane numbers. The purpose of the change to sending TS2s instead of TS1s is to advertise to the link partner that this device is ready to proceed to the next state in the state machine. It is a handshake mechanism to ensure that both devices on the link proceed through the LTSSM together. Neither device can proceed to the next state until both devices are ready. The way they advertise they are ready is by sending TS2 ordered-sets. So once a device is both sending AND receiving TS2s, it knows it can proceed to the next state because it is ready and its link partner is ready too.

### *During Polling.Configuration*

Transmitters send TS2s with Link and Lane numbers set to PAD on all detected Lanes, and they must advertise all the data rates they support, even those they don't intend to use. Also, each Lane's receiver must independently invert the polarity of its differential input pair if necessary. For an explanation of how this is done, see “Overview” on page 506. The Transmit Margin field must be reset to 000b.

# PCI Express Technology

## *Exit to "Configuration State"*

After eight consecutive TS2s with Link and Lane set to PAD are received on any detected Lanes, and at least 16 TS2s have been sent since receiving one TS2, exit to Configuration.

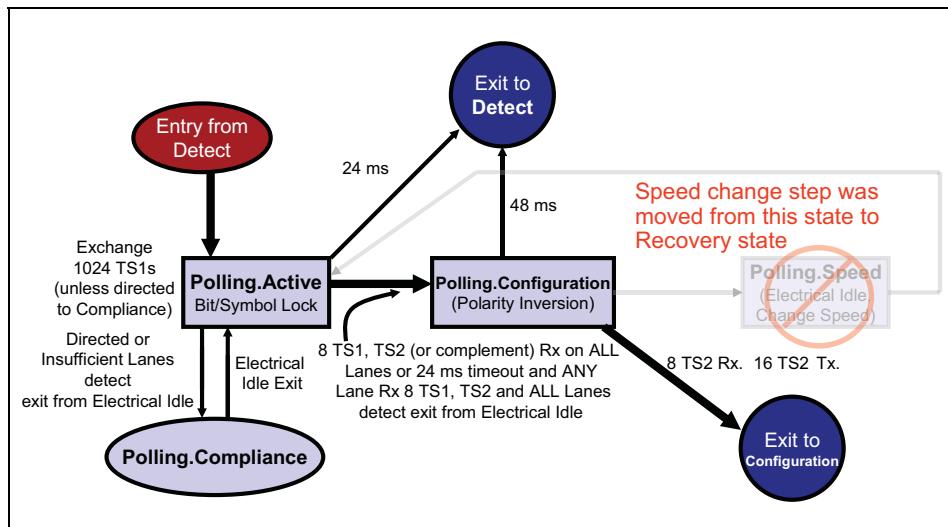
## *Exit to "Detect State"*

Otherwise, exit to Detect after a 48ms timeout.

## *Exit to Polling.Speed (Non-existent substate)*

As a historical aside, the substates of Polling have changed since the 1.0 version of the spec was released. At that time it was thought that when other speeds became available it would make sense to change to the highest available rate as soon as possible in this state. However, the advent of higher rates coincided with the realization that it would be advantageous to be able to change speeds both higher and lower during runtime for power management reasons. Going through the Polling state involves clearing a number of Link values and that makes it an unattractive path for runtime use, so the rate change stage was moved out of this state into the Recovery state. See Figure 14-10 on page 528.

Figure 14-10: Polling State Machine with Legacy Speed Change



Today, the Link always trains to 2.5 GT/s after a reset, even if other speeds are available. If higher speeds are available once the LTSSM has reached L0, then it transitions to Recovery and attempts to change to the highest commonly-supported or advertised rate. Supported speeds are reported in the exchanged TS1s

# Chapter 14: Link Initialization & Training

---

and TS2s, so that either device can subsequently decide to initiate a speed change by transitioning to the Recovery state. The spec still lists this substate but declares that it is now unreachable.

## Polling.Compliance

This substate is only used for testing and causes a Transmitter to send specific patterns intended to create near-worst-case Inter-Symbol Interference (ISI) and cross-talk conditions to facilitate analysis of the Link. Two different patterns can be sent while in this substate, the Compliance Pattern and the Modified Compliance Pattern.

**Compliance Pattern for 8b/10b.** This pattern consists of 4 Symbols that are repeated sequentially: K28.5-, D21.5+, K28.5+ and D10.2-, where (-) means negative current running disparity or CRD and (+) means positive CRD (since the CRD is forced, it's permissible to have a disparity error at the beginning of the pattern). If the Link has multiple Lanes, then four Delay Symbols (shown as D, but are really just additional K28.5 symbols) are injected on Lane 0, two before the next compliance pattern and two after the compliance pattern. Once the last Delay symbol has been sent on Lane 0, the four delay symbols are also sent on Lane 1 (again, two before the next compliance pattern and two after). This process continues until after the Delay symbols have propagated through Lane 7. Then they go back to starting on Lane 0 again as can be seen in Table 14-3 on page 529 (the compliance pattern is shaded in grey). Every group of eight lanes behaves this way. Shifting the Delay Symbols will ensure interference between adjacent Lanes and provide better test conditions.

Table 14-3: Symbol Sequence 8b/10b Compliance Pattern

Symbol	Lane 0	Lane 1	Lane 2	...	Lane 8
0	D	K28.5-	K28.5-		D
1	D	K21.5	K21.5		D
2	K28.5-	K28.5+	K28.5+		K28.5-
3	K21.5	D10.2	D10.2		K21.5
4	K28.5+	K28.5-	K28.5-		K28.5+
5	D10.2	K21.5	K21.5		D10.2

# PCI Express Technology

---

*Table 14-3: Symbol Sequence 8b/10b Compliance Pattern (Continued)*

Symbol	Lane 0	Lane 1	Lane 2	...	Lane 8
6	D	K28.5+	K28.5+		D
7	D	D10.2	D10.2		D
8	K28.5-	D	K28.5-		K28.5-
9	K21.5	D	K21.5		K21.5
10	K28.5+	K28.5-	K28.5+		K28.5+
...	...	...	...		...
16	K28.5-	K28.5-	D		K28.5-
17	K21.5	K21.5	D		K21.5
18	K28.5+	K28.5+	K28.5-		K28.5+

**Compliance Pattern for 128b/130b.** This pattern consists of the following repeating sequence of 36 Blocks:

1. The first Block consists of the Sync Header 01b and contains the unscrambled payload of 64 ones followed by 64 zeros.
2. The second Block has Sync Header 01b and contains the unscrambled payload shown in Table 14-4 on page 530 (note that the pattern repeats after 8 Lanes, and that P means the 4-bit Tx preset being used, while ~P is the bit-wise inverse of that).
3. The third Block has Sync Header 01b and contains the unscrambled payload shown in Table 14-5 on page 531 (same notes as the second Block).
4. The fourth Block is an EIEOS Block
5. 32 more Data Blocks, each containing 16 scrambled IDL Symbols (00h).

*Table 14-4: Second Block of 128b/130b Compliance Pattern*

Symbol	Lane 0	Lane 1	Lane 2	Lane 3	Lane 4	Lane 5	Lane 6	Lane 7
0	55h	FFh	FFh	FFh	55h	FFh	FFh	FFh
1	55h	FFh	FFh	FFh	55h	FFh	FFh	FFh

## Chapter 14: Link Initialization & Training

---

*Table 14-4: Second Block of 128b/130b Compliance Pattern (Continued)*

<b>Symbol</b>	<b>Lane 0</b>	<b>Lane 1</b>	<b>Lane 2</b>	<b>Lane 3</b>	<b>Lane 4</b>	<b>Lane 5</b>	<b>Lane 6</b>	<b>Lane 7</b>
2	55h	00h	FFh	FFh	55h	FFh	FFh	FFh
3	55h	00h	FFh	C0h	55h	FFh	F0h	F0h
4	55h	00h	FFh	00h	55h	FFh	00h	00h
5	55h	00h	C0h	00h	55h	E0h	00h	00h
6	55h	00h	00h	00h	55h	00h	00h	00h
7	{P,~P}							
8	00h	1Eh	2Dh	3Ch	4Bh	5Ah	69h	78h
9	00h	55h	00h	00h	00h	55h	00h	F0h
10	00h	55h	00h	00h	00h	55h	00h	00h
11	00h	55h	00h	00h	00h	55h	00h	00h
12	00h	55h	0Fh	0Fh	00h	55h	07h	00h
13	00h	55h	FFh	FFh	00h	55h	FFh	00h
14	00h	55h	FFh	FFh	7Fh	55h	FFh	00h
15	00h	55h	FFh	FFh	FFh	55h	FFh	00h

*Table 14-5: Third Block of 128b/130b Compliance Pattern*

<b>Symbol</b>	<b>Lane 0</b>	<b>Lane 1</b>	<b>Lane 2</b>	<b>Lane 3</b>	<b>Lane 4</b>	<b>Lane 5</b>	<b>Lane 6</b>	<b>Lane 7</b>
0	FFh	FFh	55h	FFh	FFh	FFh	55h	FFh
1	FFh	FFh	55h	FFh	FFh	FFh	55h	FFh
2	FFh	FFh	55h	FFh	FFh	FFh	55h	FFh
3	F0h	F0h	55h	F0h	F0h	F0h	55h	F0h
4	00h	00h	55h	00h	00h	00h	55h	00h

# PCI Express Technology

---

*Table 14-5: Third Block of 128b/130b Compliance Pattern (Continued)*

Symbol	Lane 0	Lane 1	Lane 2	Lane 3	Lane 4	Lane 5	Lane 6	Lane 7
5	00h	00h	55h	00h	00h	00h	55h	00h
6	00h	00h	55h	00h	00h	00h	55h	00h
7	{P,~P}							
8	00h	1Eh	2Dh	3Ch	4Bh	5Ah	69h	78h
9	00h	00h	00h	55h	00h	00h	00h	55h
10	00h	00h	00h	55h	00h	00h	00h	55h
11	00h	00h	00h	55h	00h	00h	00h	55h
12	FFh	0Fh	0Fh	55h	0Fh	0Fh	0Fh	55h
13	FFh	FFh	FFh	55h	FFh	FFh	FFh	55h
14	FFh	FFh	FFh	55h	FFh	FFh	FFh	55h
15	FFh	FFh	FFh	55h	FFh	FFh	FFh	55h

**Modified Compliance Pattern for 8b/10b.** The second compliance pattern adds an error status field that reports how many Receiver errors have been detected while in Polling.Compliance.

In 8b/10b mode, the original pattern is still used, but 2 Symbols are added to report the error status (2 are used instead of one to avoid interfering with the required disparity of the sequence) and 2 more K28.5 Symbols are added at the end, making the pattern 8 Symbols long altogether.

*Table 14-6: Symbol Sequence of 8b/10b Modified Compliance Pattern*

Symbol	Lane 0	Lane 1	Lane 2	...	Lane 8
0	D	K28.5-	K28.5-		D
1	D	K21.5	K21.5		D
2	D	K28.5+	K28.5+		D
3	D	D10.2	D10.2		D

## Chapter 14: Link Initialization & Training

---

Table 14-6: Symbol Sequence of 8b/10b Modified Compliance Pattern (Continued)

Symbol	Lane 0	Lane 1	Lane 2	...	Lane 8
4	K28.5-	ERR	ERR		K28.5-
5	K21.5	ERR	ERR		K21.5
6	K28.5+	K28.5-	K28.5-		K28.5+
7	D10.2	K28.5+	K28.5+		D10.2
8	ERR	K28.5-	K28.5-		ERR
9	ERR	K21.5	K21.5		ERR
10	K28.5-	K28.5+	K28.5+		K28.5-
11	K28.5+	D10.2	D10.2		K28.5+
12	K28.7-	ERR	ERR		K28.7-
13	K28.7-	ERR	ERR		K28.7-
14	K28.7-	K28.5-	K28.5-		K28.7-
15	K28.7-	K28.5+	K28.5+		K28.7-
16	K28.5-	D	K28.5-		K28.5-

The encoded error status byte contains a Receiver Error Count in ERR [6:0] that reports the number of errors seen since Pattern Lock was asserted. The “Pattern Lock” indicator is ERR bit [7], and shows when the Receiver has locked to the incoming Modified Compliance Pattern. The delay sequence is also different for this pattern, and now adds four K28.5 Symbols (shown as “D” in the table) in a row at the beginning of the sequence and four K28.7 Symbols at the end of the 8-Symbol pattern, making a total of 16 Symbols that are sent before the Delay pattern shifts to the next Lane. This pattern is illustrated in Table 14-6 on page 532. It can be seen that the delay pattern shifts to Lane 1 after 16 Symbols. As before, the basic pattern (8-Symbols now) is highlighted in grey.

**Modified Compliance Pattern for 128b/130b.** This pattern consists of a repeating sequence of 65792 Blocks as listed here:

1. One EIEOS Block
2. 256 Data Blocks of 16 scrambled IDL Symbols (00h) each.

3. 255 sets of the following sequence:
  - One SOS
  - 256 Data Blocks of 16 scrambled IDL Symbols each.

Since the payload in the Data Blocks is all zeros, the output ends up being simply the output of the scrambler for that Lane. Recall that the scrambler doesn't advance with the Sync Header bits and is initialized by the EIEOS. Since the scrambler seed value depends on the Lane number, it's important that they be understood correctly. If Link training completed earlier but then software sent the LTSSM to this substate by setting the Enter Compliance bit in the Link Control 2 register, then the Lane numbers and polarity inversions that were assigned during training are used. If a Lane wasn't active during training, or if this substate was entered in any other way, then the Lane numbers will be the default numbers assigned by the Port. Finally, note that the Data Blocks in this pattern don't form a Data Stream and don't have to follow the requirements for that (such as sending any SDS Ordered Sets or EDS Tokens).

The thoughtful reader may be wondering about the absence of error status Symbols in this sequence that are prominent in the 8b/10b sequence. As it turns out, for 128b/130b they're included inside the SOSs now. Recall that the last 2 bytes of the SOS are used to report the Receiver error count during Polling.Compliance (see "Ordered Set Example - SOS" on page 426 for more on this).

#### *Entering Polling.Compliance:*

As was the case when entering Polling.Active, the Transmit Margin field of the Link Control 2 register is used to set the Transmitter voltage range that will be in effect while in this substate.

The data rate and de-emphasis level are determined as described below. Since many of the choices about these settings depend on the Link Control 2 register fields, that register is shown in Figure 14-11 on page 536 for reference.

- If a Port only supports 2.5 GT/s, then that will be the data rate and the de-emphasis level will be -3.5dB.
- Otherwise, if this substate was entered because 8 consecutive TS1s were received with the Compliance Receive bit set to 1b and the Loopback bit cleared to 0b (bits 4 and 2 of TS1 Symbol 5), then the rate will be the highest common value for any Lane. The select\_deemphasis variable must be set to match the Selectable De-emphasis bit in TS1 Symbol 4. If the chosen rate is 8.0 GT/s, the select\_preset variable on each Lane is taken from

## Chapter 14: Link Initialization & Training

---

Symbol 6 of the consecutive TS1s. For this Gen3 rate, Lanes that didn't receive 8 consecutive TS1s with Transmitter Preset information can choose any value they support.

- Otherwise, if the Enter Compliance bit is set in the Link Control 2 register, the compliance pattern is transmitted at the data rate given by the Target Link Speed field. If the rate will be 5.0 GT/s, the select\_deemphasis variable is set if the Compliance Preset/De-emphasis field equals 0001b. If the rate will be 8.0 GT/s, the select\_preset variable of each Lane is cleared to 0b and the Transmitter must use the Compliance Preset/De-emphasis value, as long as it isn't a Reserved encoding.
- Finally, if none of the other cases are true, then the data rate, preset, and de-emphasis settings will cycle through a sequence based on the component's maximum supported speed and the number of times Polling.Compliance is entered this way. The sequence is given in Table 14-7 on page 535 and begins with Setting Number 1 the first time Polling.Compliance is entered, it increments through the list each time it's re-entered, and eventually repeats the pattern if it's re-entered more than 14 times. This provides a handy way to test all of a component's supported settings: transition to Polling.Compliance, test that setting, transition back to Polling.Active, then back to Polling.Compliance again to test the next setting. A method for a load board to cause these transitions is described in the spec, and consists of sending a 100MHz, 350mVp-p signal for about 1ms on one leg of a receiver's differential pair.

*Table 14-7: Sequence of Compliance Tx Settings*

Setting Number	Data Rate	De-emphasis	Tx Preset Encoding
1	2.5	-3.5	n/a
2	5.0	-3.5	n/a
3	5.0	-6.0	n/a
4	8.0	n/a	0000b
5	8.0	n/a	0001b
6	8.0	n/a	0010b
7	8.0	n/a	0011b
8	8.0	n/a	0100b

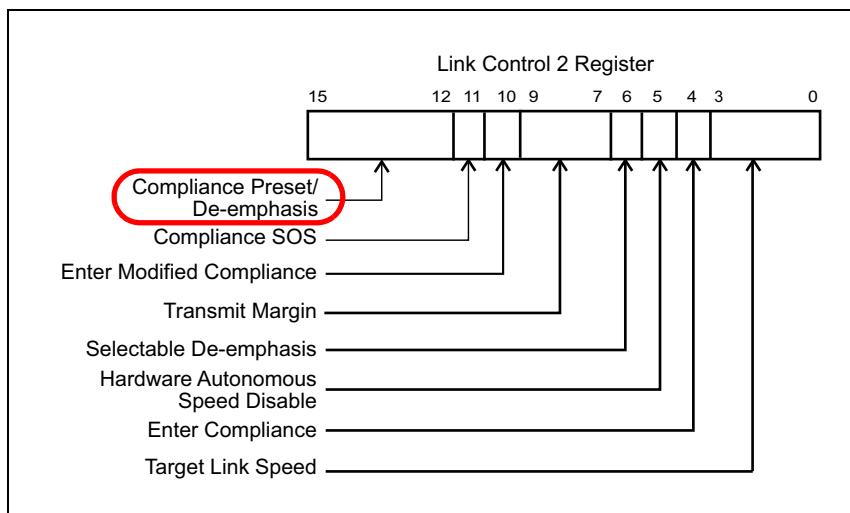
# PCI Express Technology

---

Table 14-7: Sequence of Compliance Tx Settings (Continued)

Setting Number	Data Rate	De-emphasis	Tx Preset Encoding
9	8.0	n/a	0101b
10	8.0	n/a	0110b
11	8.0	n/a	0111b
12	8.0	n/a	1000b
13	8.0	n/a	1001b
14	8.0	n/a	1010b

Figure 14-11: Link Control 2 Register



If the data rate won't be 2.5 GT/s, then:

- If any TS1s were sent during Polling.Active, the Transmitter must send either one or two consecutive EIOSs before going into Electrical Idle.
- If no TS1s were sent in Polling.Active, the transmitter enters Electrical Idle without sending any EIOSs.
- The Electrical Idle period must be >1ms and <2ms. During this time, the data rate is changed to the new speed and stabilized. If the rate will be 5.0 GT/s, the de-emphasis level is given by the select\_deemphasis variable

## **Chapter 14: Link Initialization & Training**

---

(0b = -3.5dB, 1b = -6.0 dB). If the rate will be 8.0 GT/s, then the select\_preset variable gives the transmitter presets to use.

### *During Polling.Compliance:*

Once the data rate and de-emphasis or preset values have been determined, the following rules will apply:

**Compliance Pattern.** If entry was not due to the Compliance Receive bit set and Loopback bit cleared in the TS Ordered Sets and was not due to both the Enter Compliance and Enter Modified Compliance bits being set in the Link Control 2 register, then Transmitters send the compliance pattern on all detected Lanes.

### *Exit to "Polling.Active"*

If any of these conditions are true:

- a) Electrical Idle exit is detected at the Receiver of any detected Lane and the Enter Compliance bit is cleared (0b).  
The spec notes that the stipulation “any Lane” supports the Load Board usage model described earlier to allow the device to cycle through all the supported test cases.
- b) The Enter Compliance bit has been cleared (0b) since Polling.Compliance was entered.
- c) For an Upstream Port, the Enter Compliance bit is set (1b) and EIOS has been detected on any Lane. This condition clears the Enter Compliance bit (0b).

If the data rate was not 2.5 GT/s or the Enter Compliance bit was set during entry to Polling.Compliance, the Transmitter sends 8 consecutive EIOSs and goes to Electrical Idle before transitioning to Polling.Active. During the Electrical Idle time the Port changes to 2.5 GT/s and stabilized for a time between 1ms and 2ms.

Sending multiple EIOSs helps ensure that the Link partner will detect at least one and exit Polling.Compliance when the Enter Compliance register bit was used for entry

**Modified Compliance Pattern.** If Polling.Compliance was entered because TS1s directed it, and either the Compliance Receive bit was set and Loopback bit was cleared or both Enter Compliance and Enter Modified Compliance bits were set in Link Control 2 register then send the Modified Compliance Pattern on all detected Lanes with the error status Symbol cleared to all zeroes.

# PCI Express Technology

---

If the rate is 2.5 or 5.0 GT/s, each Lane indicates a successful lock on the incoming pattern by looking for one instance of the Modified Compliance Pattern and then setting the Pattern Lock bit in the Modified Compliance Pattern that it sends back (bit 7 of the 8-bit error status Symbol).

- The error status Symbols cannot be used in the locking process because they don't have meaning if the Link partner isn't already locked and therefore their meaning can be undefined.
- An instance of the pattern is defined to be the sequence of 4 Symbols described earlier: K28.5, D21.5, K28.5, and D10.2 or the complement of these Symbols (meaning the polarity is inverted).
- The device under test must set the Pattern Lock bit in the Modified Compliance Patterns it sends within 1ms of receiving the Modified Compliance Pattern from the Link partner.
- Any Receiver errors on a Lane increment that Lane's error count by 1, and it saturates when the count reaches 127 (doesn't go higher or wrap around).

If the rate is 8.0 GT/s

- The Error\_Status field is set to 00h on entry to this substate.
- The device under test must set the Pattern Lock bit in the Modified Compliance Patterns it sends within 4ms of receiving the Modified Compliance Pattern from the Link partner.
- Each Lane independently sets Pattern Lock when it achieves Block Alignment. After that, Symbols in Data Blocks are expected to be IDLs (00h) and any mismatched Symbols increment the count by 1. The Receiver Error Count saturates at 127, and is sent in the last 2 Symbols of the SOS's included in this pattern.
- The scrambling requirements are applied as usual to the Modified Compliance Pattern: the seed value is set per Lane, an EEOS initiates the LFSR, and SOS's don't advance the LFSR.
- The spec notes that devices should wait long enough before acquiring Block alignment to ensure that their Receivers have stabilized and won't see any bit slips. It even mentions that devices might want to re-validate their Block alignment before setting the Pattern Lock bit.

*Exit to "Polling.Active"*

If the Enter Compliance bit was set (1b) on entry to Polling.Compliance and either the Enter Compliance bit has been cleared (0b), or it's an Upstream Port and received an EIOS on any Lane. This also causes its Enter Compliance bit to be cleared (0b).

## Chapter 14: Link Initialization & Training

---

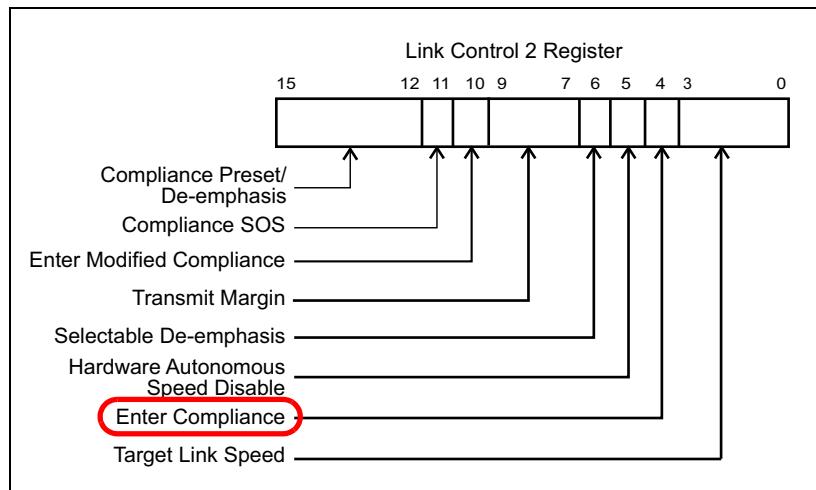
If the data rate was not 2.5 GT/s or the Enter Compliance bit was set during entry to Polling.Compliance, the Transmitter sends 8 consecutive EIOSs and goes to Electrical Idle before transitioning to Polling.Active. During the Electrical Idle time the Port changes to 2.5 GT/s and -3.5dB de-emphasis, and this time must be between 1ms and 2ms.

Sending multiple EIOSs helps ensure that the Link partner will detect at least one and exit Polling.Compliance when the Enter Compliance register bit was used for entry.

*Exit to “Detect State”*

If the Enter Compliance bit in the Link Control 2 register is cleared (0b) and the device is directed to exit this substate.

*Figure 14-12: Link Control 2 Register’s “Enter Compliance” Bit*



---

## Configuration State

Initially, the Configuration state performs Link and Lane Numbering at the 2.5 GT/s rate; however, provisions exist that allow the 5 GT/s and 8 GT/s devices to also enter the Configuration state from the Recovery state. The transition from Recovery to Configuration is done primarily for making dynamic changes in the link width of multi-lane devices. The dynamic changes are supported for the 5 GT/s and 8 GT/s devices only. Consequently, the detailed state transitions for these devices appear in the detailed Configuration Substate descriptions beginning on page 552.

## Configuration State — General

The main goal of this state is to discover how the Port has been connected and assign Lane numbers for it. For example, 8 Lanes may be available but only 2 are active, or perhaps the Lanes can be split into multiple Links, such as two x4 Links. Unlike the other states, Ports have defined roles that depend on whether they are facing upstream or downstream. For that reason, the description of these substates is grouped into the behavior for Downstream Lanes and for Upstream Lanes. The Downstream Port (port that transmits downstream) plays the “leader” role on this Link to walk through the rest of the states in the link initialization process. The Upstream Port (port that transmits upstream) plays the “follower” role. The leader, or Downstream Port, will specify the Link and Lane numbers to the Upstream Port, and the Upstream Port will simply reply with the same values it was told, unless there is a conflict, as we will see in this section. The Link and Lane numbers are reported in the fields of the TS1s exchanged during this time, as shown again in Figure 14-13 on page 540. These fields contain PAD symbols as a placeholder until actual values are assigned.

Figure 14-13: Link and Lane Number Encoding in TS1/TS2

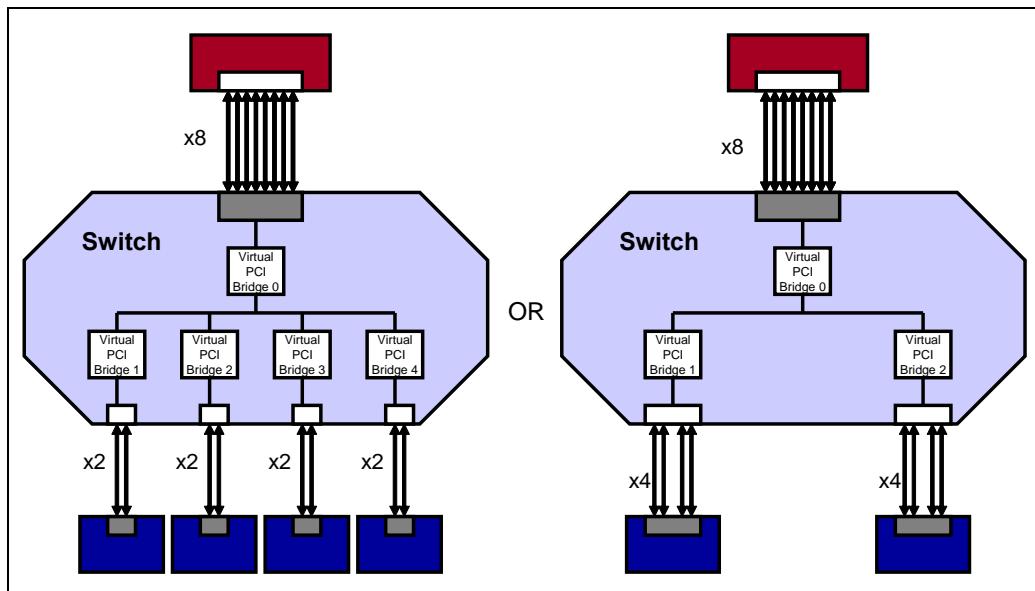
0	COM	K28.5
1	Link #	0 - 255 = D0.0 - D31.7, PAD = K23.7
2	Lane #	0 - 31 = D0.0 - D17.1, PAD = K23.7
3	# FTS	# of FTSs required by Receiver for L0s recovery
4	Rate ID	Bit 1 must be set, indicates 2.5 GT/s support
5	Train Ctl	
6	TS ID or	
9	EQ Info	
10		
15	TS ID	TS1 Identifier = D10.2 TS2 Identifier = D5.2

## Designing Devices with Links that can be Merged

A designer chooses how many Lanes to implement on a given Link based on performance and cost requirements. Narrow Links may optionally be able to combine into a wider Link, and a wide Link can optionally be split into multiple narrower Links. Figure 14-14 on page 541 shows a Switch with one Upstream Port and four x2 Downstream Ports. In this example, they can also be grouped into two x4 Links. As a reminder, the spec requires that every Port must also support operating as a x1 Link.

As seen on the left side of the figure, the switch internally consists of one upstream logical bridge and four downstream logical bridges. One bridge is required for each Port, so supporting 4 Downstream Ports requires 4 downstream bridges. However, if the Ports are combined as shown on the right side of the diagram, then some of the bridges simply go unused. During Link Training, the LTSSM of each Downstream Port determines which of the supported connection options is actually implemented.

Figure 14-14: Combining Lanes to Form Wider Links (Link Merging)



## Configuration State — Training Examples

### Introduction

In the Configuration state, the Link and Lane numbering process is initiated by a Downstream Port, the “leader,” (e.g., Root Port or Switch Downstream Port). Endpoints and switch Upstream Ports don’t initiate, but respond. They are the “follower.” Let’s now consider some examples to make the concepts easier to understand.

### Link Configuration Example 1

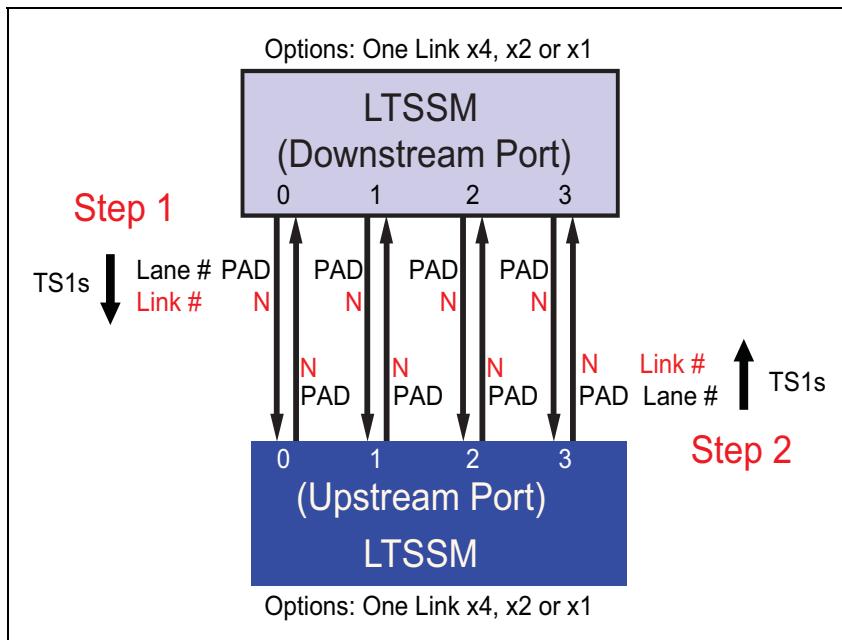
The devices shown in Figure 14-15 on page 543 both support a single Link that implements lane sizes of x4, x2, or x1. The Lane number assignments are fixed by the device internally and must be sequential starting from zero. The physical Lane numbers are shown within the device box and the reported, or logical, Lane numbers are reported by the TS Ordered Sets. Usually, these will be the same, but not in every case.

#### Link Number Negotiation.

1. Since only one Link is possible in this example, the Downstream Port (the Port that transmits downstream) sends TS1s using the same Link Number,  $N$ , for all the Lanes and PAD for the Lane Numbers.
2. In this Configuration state, the Upstream Port starts out sending TS1s with PAD in the Link and Lane number fields, but upon receiving the TS1s from the Downstream Port with the non-PAD Link number, the Upstream Port responds with TS1s on all connected Lanes that reflect the same Link Number  $N$  and PAD for the Lane Number field. Based on this response, the Downstream LTSSM recognizes that four Lanes responded and used the same Link number as is being sent, so all 4 Lanes will be configured as one Link. The Link Number itself is an implementation-specific value that isn’t stored in any defined configuration register and isn’t related to the Port Number or any other value.

## Chapter 14: Link Initialization & Training

Figure 14-15: Example 1 - Steps 1 and 2



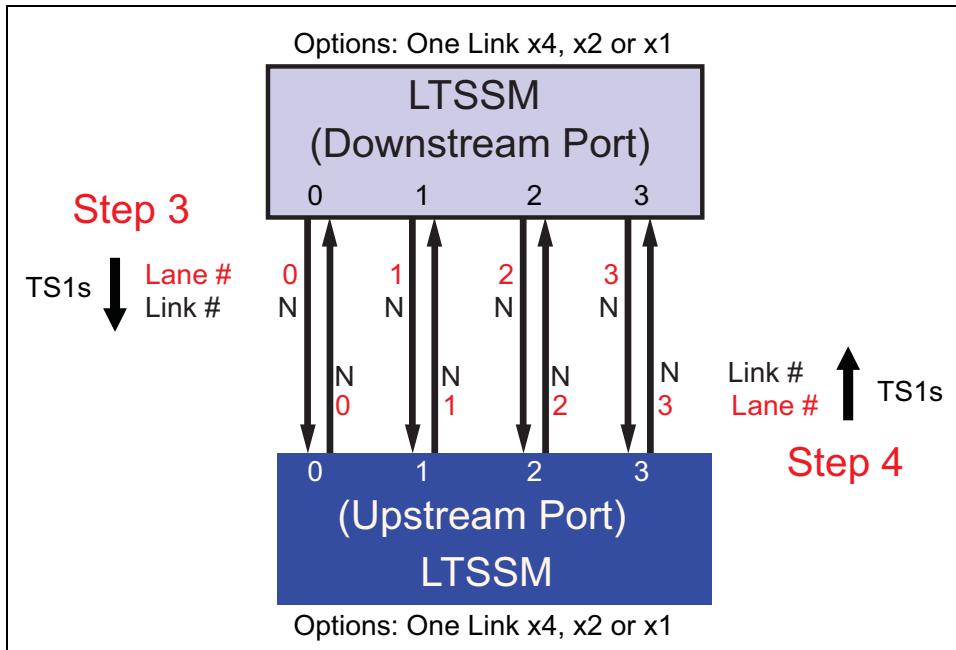
### Lane Number Negotiation.

3. The Downstream Port now begins to send TS1s with the same Link Number but assigns Lane Numbers of 0, 1, 2 and 3 to the connected Lanes, as shown in Figure 14-16 on page 544.
4. In response to seeing non-PAD Lane numbers coming in, the Upstream Port will verify that the incoming Lane numbers match the Lane numbers they are received on. In this example, the Lanes of the Downstream and Upstream Ports are connected correctly. Because all the Lane numbers match, the Upstream Port advertises its Lane numbers in the TS1s it is sending as well. When the Downstream Port sees non-PAD Lane numbers in response, it compares the incoming numbers to the values it's sending. If they match, all is well but, if not, then other steps will need to be taken. If some but not all Lane numbers match, then the Link width may be adjusted accordingly. If the Lanes are reversed, then the optional Lane Reversal feature will be needed. Because it's optional, it's possible that the Lanes have been reversed but neither device is capable of correcting it. This would be a dramatic board design error because it is possible the Link cannot be configured for operation in this case.

# PCI Express Technology

---

Figure 14-16: Example 1 - Steps 3 and 4

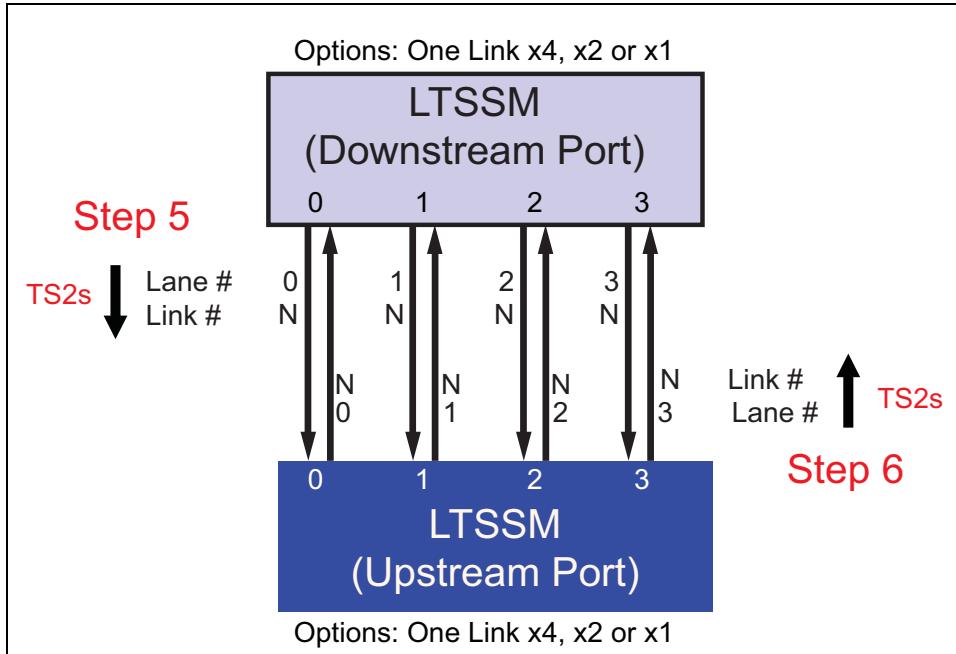


## Confirming Link and Lane Numbers.

5. Since the transmitted and received Link and Lane numbers matched on all the Lanes, the Downstream Port indicates it is ready to conclude this negotiation and proceed to the next state, L0, by sending TS2 Ordered Sets with the same Link and Lane numbers.
6. Upon receiving TS2s with the same Link and Lane numbers, the Upstream Port also indicates its readiness to leave the Configuration state and proceed to L0 by sending TS2s back. This is shown in Figure 14-17 on page 545.
7. Once a Port receives at least 8 TS2s and transmits at least 16, it sends some logical idle data and then transitions to L0.

## Chapter 14: Link Initialization & Training

Figure 14-17: Example 1 - Steps 5 and 6



### Link Configuration Example 2

Another example that should be covered is of a Device with 4 Downstream Lanes that is capable of being configured as a single x4 Link or a combination of two x2 Links or four x1 Links. So even a configuration of one x2 Link and two x1 Links would be just fine. An example of this type of Device can be seen in Figure 14-18 on page 546.

If all four Lanes have detected a receiver and made it to the Configuration state, there are a number of connection possibilities:

- One x4 Link
- Two x2 Links
- One x2 Link and two x1 Links
- Four x1 Links

One example method defined in the spec to determine which of the configurations are implemented is described below.

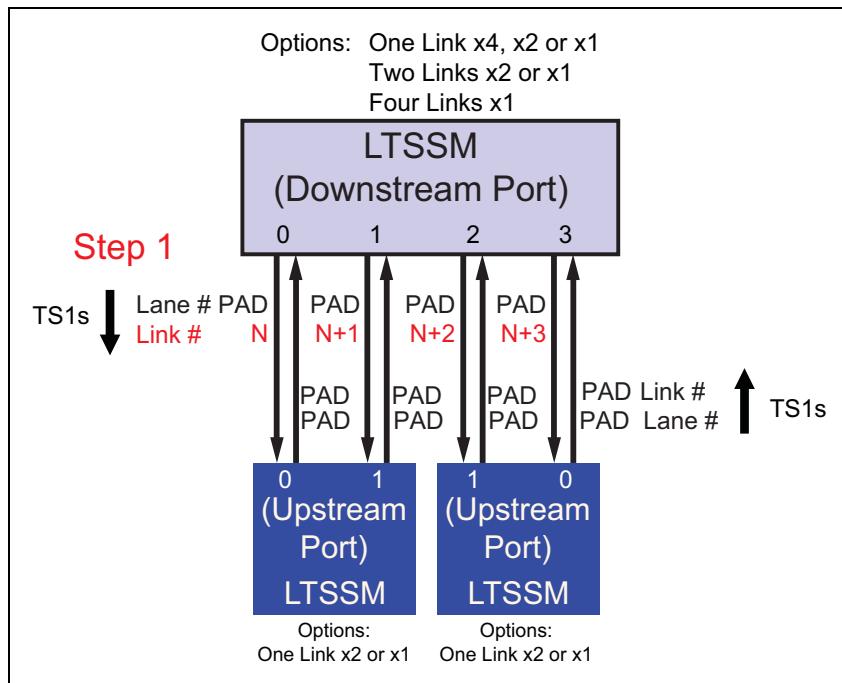
# PCI Express Technology

---

## Link Number Negotiation.

1. In this example method, the Downstream Port begins by advertising a unique Link number on each Lane. Lane 0 advertises a Link number of N, Lane 1 advertises a Link number of N+1, etc. as shown in Figure 14-18 on page 546. These Link numbers are just examples, and they do not have to be sequential. Also, it is important to remember that the Downstream Port does not know what it is connected to and it is this process where the Port is trying to determine the connections for each Lane.

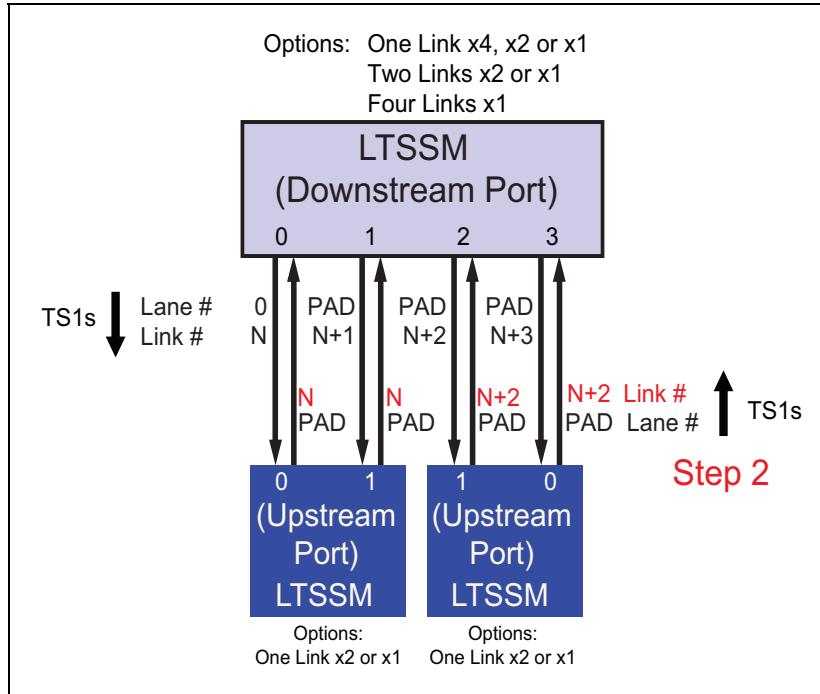
Figure 14-18: Example 2 - Step 1



2. Upon receiving the returned TS1s, the Downstream Port recognizes two things: all four Lanes are working and they are connected to two different Upstream Ports. This means there will actually be *two* Downstream Ports. Each Downstream Port will have its own Lane 0 and Lane 1 as shown in Figure 14-20 on page 548.

## Chapter 14: Link Initialization & Training

Figure 14-19: Example 2 - Step 2



### Lane Number Negotiation.

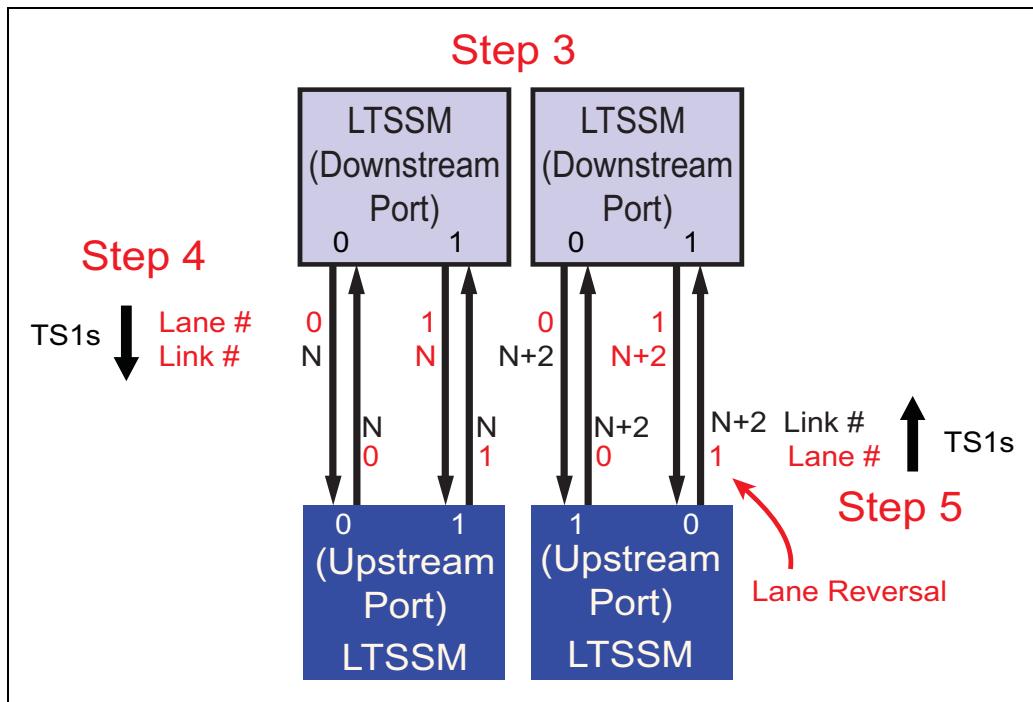
3. The process continues now for each Link independently but they'll take the same steps as before to determine the Lane numbers: the Downstream Ports will advertise their Lane numbers in the TS1s. It is also important to note that the Downstream Ports begin advertising the single returned Link number for all Lanes of the Link. The Link on the left is advertising a Link number of N for both Lanes and the Link on the right is advertising N+2.
4. In this example, the Lane numbers of the Link on the left match between the Downstream and Upstream Port. However, for the Link on the right, the Lane numbers of the Downstream Port are reversed from the connected Upstream Port. The Upstream Port realizes this and if it supports Lane Reversal, it will implement that internally and reply back with the same Lane numbers that were advertised by the Downstream Port, as shown in Figure 14-20. If the Upstream Port did not support Lane Reversal, it would have advertised its own Lane numbers in

# PCI Express Technology

the returned TS1s and then the Downstream Port would have realized the issue and had a chance to implement Lane Reversal.

5. Lane Reversal can optionally be handled by either Port. If the Upstream Port detects this case and supports Lane Reversal, it simply makes the Lane assignment change internally and returns TS1s with the proper Lane numbers. As a result, the Downstream Port is unaware that there was ever an issue. If the Upstream Port is unable to handle Lane Reversal though, then the Downstream Port will see the incoming Lane numbers in reverse order. If it supports Lane Reversal, it will then correct the numbering and begin sending TS2s with the new Lane numbers.

Figure 14-20: Example 2 - Steps 3, 4 and 5



## Confirming Link and Lane Numbers.

6. The Downstream Ports receive the TS1s with the Link and Lane numbers that match what was advertised so each Port, independently, starts sending TS2s as a notification that it is ready to proceed to the L0 state with the negotiated settings.

## **Chapter 14: Link Initialization & Training**

---

7. The Upstream Ports receive the TS2s with no Link and Lane number changes and start transmitting TS2s in return with the same values.
8. Once each Port receives at least 8 TS2s and transmits at least 16 TS2s, it sends some logical idle data and then transitions to L0.  
The Upstream Port of the Link on the right is implementing Lane Reversal internally.

### **Link Configuration Example 3: Failed Lane**

Finally, let's consider what happens if one of the Lanes isn't working properly. Consider an example in which Lane 2 of the Upstream Port is not functioning well as shown in Figure 14-21 on page 550. It's important to note that the Lane isn't physically broken because if it were it wouldn't have detected a Receiver and wouldn't be considered for inclusion in the Link. However, even though the Lane is attached, either the Transmitter or Receiver (or both) of Lane 2 on the Upstream Port is not getting the job done.

In cases like this, it is likely that the link training process will take considerably longer because most of the state transitions wait to proceed to the next state until ALL Lanes are ready for the next state, OR if a subset of Lanes are ready and a timeout condition has occurred.

The steps below indicate a way this situation could be handled when transitioning through the substates of the Configuration state machine.

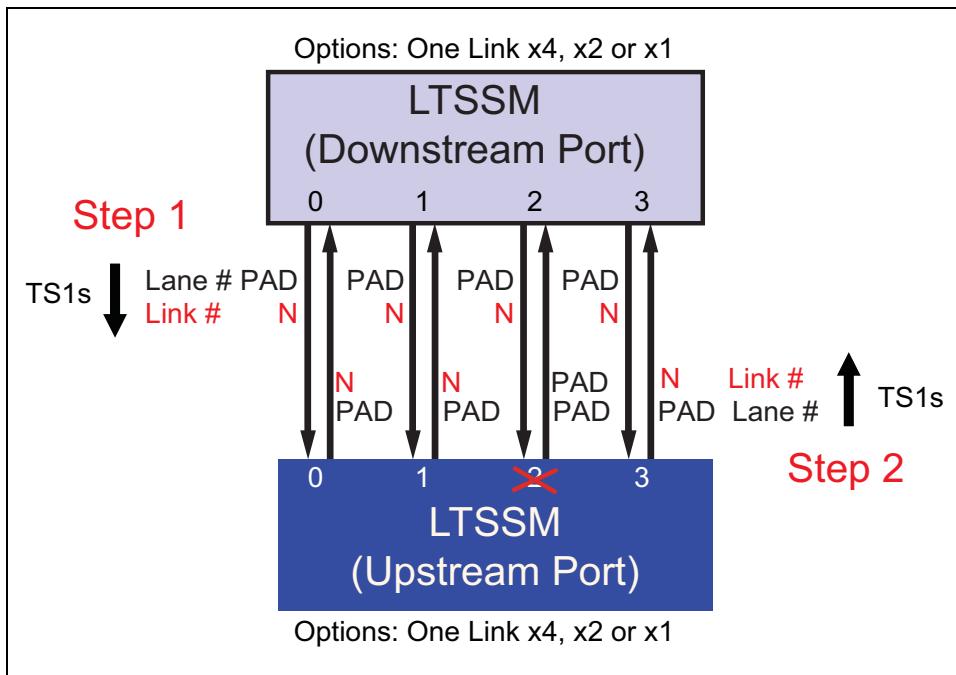
#### **Link Number Negotiation.**

9. Even though the Lane 2 Receiver on the Upstream Port is having issues, the Downstream Port is going to take the same process upon entering the Configuration state. The Downstream Port sends TS1s on all Lanes with the Link number N and with the Lane number set to PAD.
10. Lanes 0, 1 and 3 all received the TS1s with the non-PAD Link number, so those Lanes send TS1s back to the Downstream Port. However, Lane 2 of the Upstream Port did not successfully receive the TS1s with the non-PAD Link number, so its Transmitter continues sending TS1s with PAD in the Link and Lane number fields as shown in Figure 14-21 on page 550.

# PCI Express Technology

---

Figure 14-21: Example 3 - Steps 1 and 2

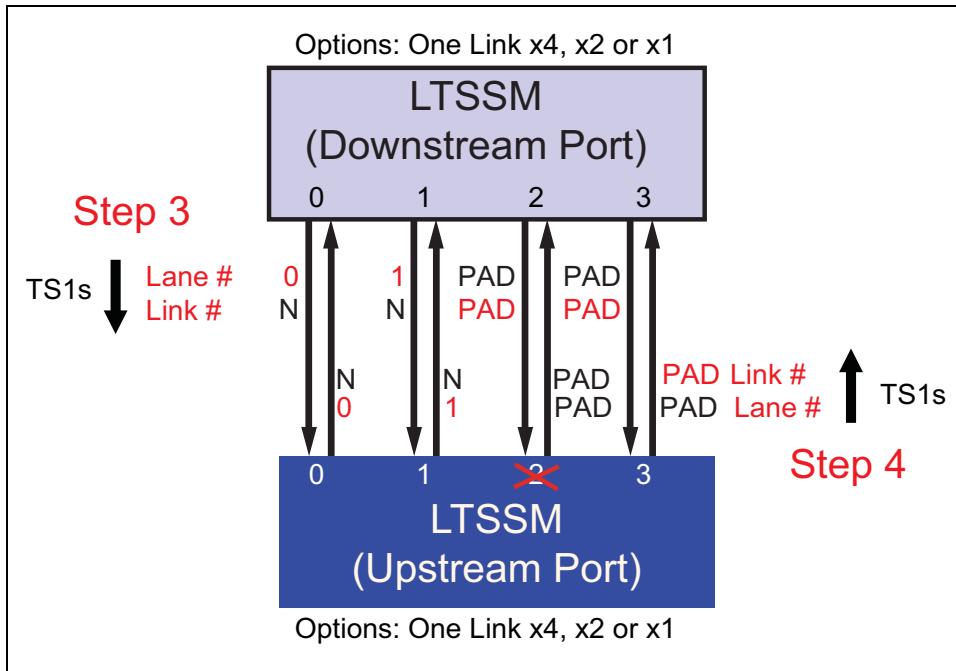


## Lane Number Negotiation.

11. Once the Downstream Port has received the TS1s with the same Link number on Lanes 0, 1 and 3, it waits until the required timeout period hoping that Lane 2 will start working. When that doesn't happen, the Downstream Port realizes that it will only be able to train as a x2 Link. After accepting this fact, the Downstream Port will advertise its Lane numbers for Lanes 0 and 1, but Lanes 2 and 3 go back to send PADs in the Link and Lane number fields.
12. When the Upstream Port receives the TS1s on Lanes 0 and 1 with the advertised Lane numbers and it sees that Lane 3 has gone back to receiving PAD TS1s, it advertises its Lane number for Lanes 0 and 1 but all the other Lanes start (or continue) sending TS1s with PAD set in both the Lane and Link number fields as shown in Figure 14-22 on page 551.

## Chapter 14: Link Initialization & Training

Figure 14-22: Example 3 - Steps 3 and 4



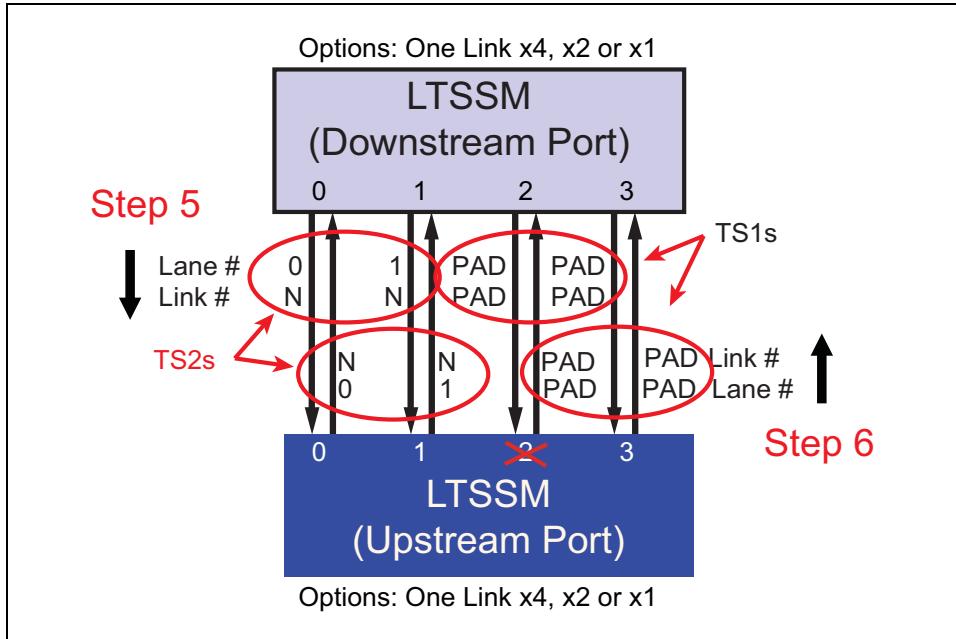
### Confirming Link and Lane Numbers.

13. Since the transmitted and received Link and Lane numbers matched on Lanes 0 and 1, the Downstream Port indicates it is ready to conclude this negotiation and proceed to the next state, L0, by sending TS2 Ordered Sets with the same Link and Lane numbers on these Lanes. The other Lanes continue sending TS1s with PAD for both the Link and Lane numbers.
14. Upon receiving TS2s with the same Link and Lane numbers on Lanes 0 and 1, the Upstream Port also indicates its readiness to leave the Configuration state and proceed to L0 by sending TS2s back on these Lanes. The other Lanes continue sending TS1s with PAD for both the Link and Lane numbers. This is shown in Figure 14-23 on page 552.

# PCI Express Technology

---

Figure 14-23: Example 3 - Steps 5 and 6



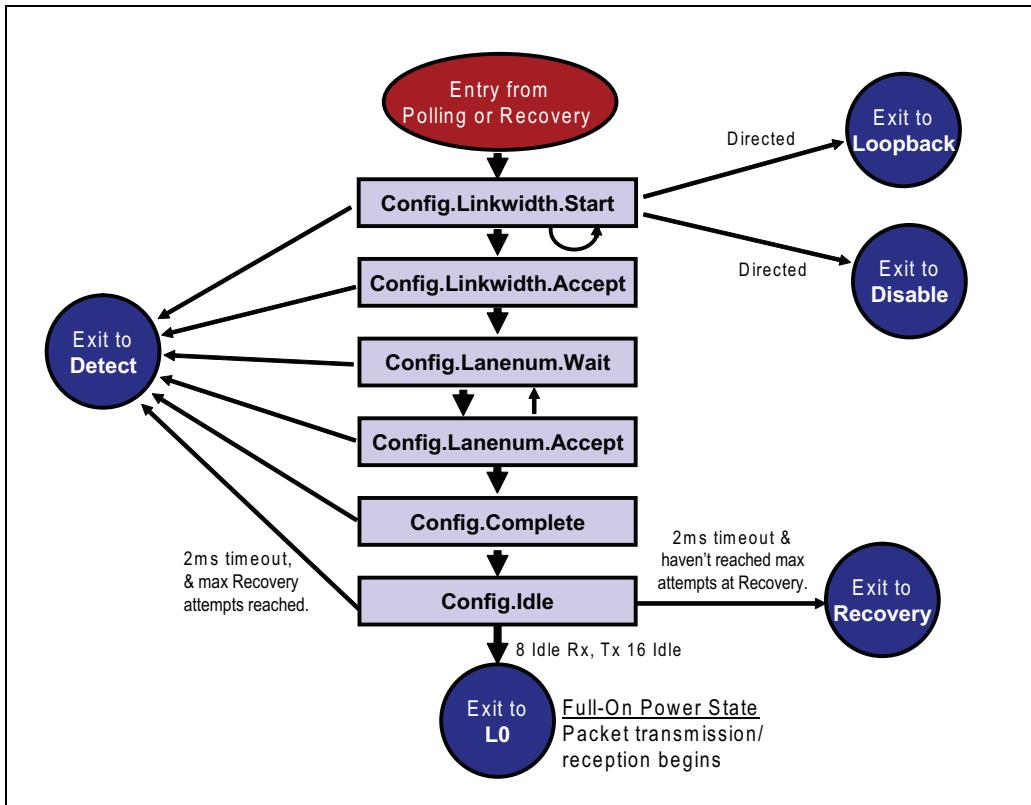
Once a Port receives at least 8 TS2s and transmits at least 16, it sends some logical idle data and those Lanes transitions to L0. The other Lanes, Lanes 2 and 3 in this example, transition to Electrical Idle until the next time the link training process is initiated at which point those Lanes will attempt the training process like normal.

---

## Detailed Configuration Substates

A detailed explanation of each substate is presented here to cover all the substates of Configuration, as shown in Figure 14-24 on page 553. The Configuration Substates should be easier to follow, given the Link Training examples discussed previously.

Figure 14-24: Configuration State Machine



## Configuration.Linkwidth.Start

This substate is entered after either the normal completion of the Polling state (as described in “Polling.Configuration” on page 527), or if the Recovery state finds that Link or Lane numbers have changed since the last time they were assigned and thus the recovery process can’t finish normally (as described in the “Recovery State” on page 571).

### Downstream Lanes.

*During Configuration.Linkwidth.Start*

The Downstream Port is now the leader on this Link and sends TS1s with a non-PAD link number on all active Lanes (as long as LinkUp is

not set and upconfiguration of the Link width is not taking place). In the TS1s, the Link number field is changed from PAD to a number while the Lane number remains PAD. The only constraint on the value of the Link numbers in the spec is that they must be unique for each possible Link if multiple Links are supported. For example, a x8 Link would have the same Link number on all 8 Lanes, but if it could also be configured as two x4 Links, both groups of 4 Lanes would be assigned different Link numbers, such as 5 for one group and 6 for the other. The values are local to the Link partners and there's no need for software to track them or try to make them unique throughout the system.

If the upconfigure\_capable bit is set to 1b, these TS1s will also be sent on any inactive Lanes that received two consecutive TS1s with Link and Lane numbers set to PAD.

- When entering this substate from Polling, any Lane that detected a Receiver is considered active.
- When entering from Recovery, any Lane that was part of the Link after going through Configuration.Complete is considered an active Lane.
- All supported data rates must be advertised in the TS1s, even if the Port doesn't intend to use them.

**Crosslinks.** For cases where LinkUp = 0b and the optional crosslink capability is supported, all Lanes that detected a Receiver must send a minimum of 16 to 32 TS1s with a non-PAD Link number and PAD Lane number. After that, the port will evaluate what it is receiving to see if a crosslink is present.

**Upconfiguring the Link Width.** If LinkUp = 1b and the LTSSM wants to upconfigure the Link, TS1s with Link and Lane numbers set to PAD are sent on the currently active Lanes, the inactive Lanes it intends to activate, and the Lanes that have seen incoming TS1s. When the Lanes have received two consecutive TS1s coming back, or after 1ms, the Link number is assigned a value in the TS1s being sent.

- If activating an inactive Lane, the Transmitter must wait for the Tx common mode voltage to settle before exiting Electrical Idle and sending TS1s.
- Link numbers must be the same for Lanes that will be grouped into a Link. The numbers can only be different for groups of Lanes that are capable of acting as a unique Link.

*Exit to "After a 24ms timeout if none of the other conditions are true."*

Any Lanes that previously received at least one TS1 with Link and Lane

## **Chapter 14: Link Initialization & Training**

---

number of PAD now receive two consecutive TS1s with a non-PAD Link number that matches a transmitted Link number and Lane numbers are still PAD will exit to the Configuration.Linkwidth.Accept sub-state.

### *Exit to "Configuration.Linkwidth.Start"*

If the first set of received TS1s for this substate have a non-PAD Link number then it's understood that a crosslink is present and the Link neighbor is also behaving as a Downstream Port. To handle this situation, the Downstream Lanes are changed to Upstream Lanes and a random crosslink timeout is chosen. The next substate will be the same Configuration.Linkwidth.Start again but the Lanes will now behave as Upstream Lanes.

This supports the optional behavior when both Link partners behave as Downstream Ports. The solution for this situation is to change both to Upstream Ports and assign each a random timeout that, when it expires, changes it to a Downstream Port. Since the timeouts won't be the same, eventually one Port is seen as Downstream while the other is seen as Upstream and then the training can go forward. The timeout must be random so that even if two of the same devices are connected any possible deadlock will eventually be broken.

If crosslinks are supported, receiving a sequence of TS1s that first have a Link number of PAD and later have a non-PAD Link number that matches the transmitted Link number is valid only if the sequence wasn't interrupted by a TS2.

### *Exit to "Disable State"*

If the Port is instructed by a higher layer to send TS1s or TS2s with the Disable Link bit asserted on all detected Lanes. Normally, the Downstream Port will initiate this but, for the optional crosslink case, it could become an Upstream Port instead and then Disabled will be the next state if 2 consecutive TS1s are received with the Loopback bit set.

### *Exit to "Loopback State"*

If the loopback-capable Transmitter is instructed by a higher layer to send TS Ordered Sets with the Loopback bit asserted, or if Lanes that are sending TS1s receive 2 consecutive TS1s with the Loopback bit set. Whichever Port sends the TS1s with the bit set will become the Loopback master, while the Port that receives them will become the Loopback slave.

# PCI Express Technology

---

*Exit to "Detect State"*

After a 24ms timeout if none of the other conditions are true.

## Upstream Lanes.

*During Configuration.Linkwidth.Start*

The Upstream Port is now the follower on this Link and goes back to sending TS1 ordered-sets with PAD set for the Link and Lane number fields. It will continue to do this until it begins receiving TS1s with a non-PAD Link number from the Downstream Port (leader).

The Upstream Port sends TS1s with Link and Lane values of PAD on a) all active Lanes, b) the Lanes it wants to upconfigure and, c) if upconfigure\_capable is set to 1b, on each of the inactive Lanes that have received two consecutive TS1s with Link and Lane numbers set to PAD while in this substate.

- When entering this substate from Polling, any Lane that detected a Receiver is considered active.
- When entering from Recovery, any Lane that was part of the Link after going through Configuration.Complete is considered an active Lane. If the transition wasn't caused by an LTSSM timeout, the Transmitter must set the Autonomous Change bit (Symbol 4, bit 6) to 1b in the TS1s being sent in the Configuration state if it does, in fact, plan to change the Link width for autonomous reasons.
- All supported data rates must be advertised in the TS1s, even if the Port doesn't intend to use them.

**Crosslinks.** For cases where LinkUp = 0b and the optional crosslink capability is supported, all Lanes that detected a Receiver must send a minimum of 16 to 32 TS1s with Link and Lane values of PAD. After that, the port will evaluate what it is receiving to see if a crosslink is present.

*Exit to "After a 24ms timeout if none of the other conditions are true."*

If *any* Lanes receive two consecutive TS1s with non-PAD Link number and PAD Lane number, this port transitions to the Configuration.Linkwidth.Accept substate where one of the received Link numbers is selected for those Lanes and TS1s are sent back with that Link number and a PAD Lane number, on *all* the Lanes that received TS1s with a non-PAD Link number. Any left-over Lanes that detected a Receiver but no Link number must send TS1s with Link and Lane numbers set to PAD.

- If upconfiguring the Link, the LTSSM waits until it receives two consecutive TS1s with a non-PAD Link number and PAD Lane number on either a) all the inactive Lanes it wants to activate, or b) on any

## **Chapter 14: Link Initialization & Training**

---

Lane 1ms after entering this substate, whichever is earlier. After that, it sends TS1s with the selected Link number along with PAD Lane numbers.

- To avoid configuring a Link smaller than necessary, it's recommended that a multi-Lane Link that sees an error or loses Block Alignment on some Lanes delay this Receiver evaluation. For 8b/10b encoding, it should wait at least two more TS1s, while for 128b/130b encoding it should wait for at least 34 TS1s, but never more than 1ms in any case.
- After activating an inactive Lane, the Transmitter must wait for the Tx common mode voltage to settle before exiting Electrical Idle and sending TS1s.

### *Exit to "Configuration.Linkwidth.Start"*

After a crosslink timeout, send 16 to 32 TS2s with Link and Lane values of PAD. The Upstream Lanes change to Downstream Lanes and the next substate will be the same Configuration.Linkwidth.Start again but this time the Lanes behave as Downstream Lanes. For the case of two Upstream Ports connected together, this optional behavior allows one of them to eventually take the lead as a Downstream Port.

### *Exit to "Disable State"*

If either of the following is true:

- Any Lanes that are sending TS1s also receive TS1s with the Disable Link bit asserted.
- The optional crosslink is supported and either all Lanes that are sending and receiving TS1s receive the Disable Link bit in two consecutive TS1s, or else a crosslink Port is directed by a higher Layer to assert the Disable bit in its TS1s and TS2s on all Lanes that detected a Receiver.

### *Exit to "Loopback State"*

If a loopback-capable Transmitter is directed by a higher Layer to send TS Ordered Sets with the Loopback bit asserted or all Lanes that are sending and receiving TS1s receive 2 consecutive TS1s with the Loopback bit set. Whichever Port sends the TS1s with the bit set will become the Loopback master, while the Port that receives them will become the Loopback slave.

### *Exit to "Detect State"*

After a 24ms timeout if none of the other conditions are true.

## Configuration.Linkwidth.Accept

At this point, the Upstream Port is now sending back TS1 ordered-sets on all its Lanes with the same Link number. The Link number originated from the Downstream Port, and the Upstream Port is simply reflecting that value back on all its Lanes. Now the Downstream Port knows the Link width (number of Lanes receiving the same Link number) and it must start advertising the Lane numbers. So the leader (Downstream Port) continues sending TS1s, but now with the actual Lane numbers designated instead of PAD. Also, all these TS1s will have the same Link number. The detailed behavior for the Downstream and Upstream Lanes are outlined below:

### Downstream Lanes

#### *During Configuration.Linkwidth.Accept*

The Downstream Port will now initiate Lane numbers. If a Link can be formed from at least one group of Lanes that all receive two consecutive TS1s and all see the same Link number, then TS1s are sent that keep that same Link number but now assign unique, non-PAD Lane numbers as well.

#### *Exit to "Configuration.Lanenum.Wait"*

The Downstream Port does not stay in the Configuration.Linkwidth.Accept substate very long. Once it has received the necessary TS1s from the Upstream Port indicating the Link width, it updates any internal state info that is required, starts sending TS1s with non-PAD Lane numbers, as indicated above, and immediately transitions to Configuration.Lanenum.Wait to await Lane Number confirmation from the Upstream Port.

### Upstream Lanes

#### *During Configuration.Linkwidth.Accept*

The Upstream Port transmits TS1s where one of the received Link numbers is selected and sent back in the TS1s on *all* the Lanes that received TS1s with a non-PAD Link number. Any left-over Lanes that detected a Receiver but no Link number must send TS1s with Link and Lane numbers set to PAD.

#### *Exit to "Configuration.Lanenum.Wait"*

The Upstream Port must respond to the Lane numbers proposed to it by the Link neighbor. If a Link can be formed using Lanes that sent a non-PAD Link number on their TS1s and received two consecutive TS1s with the same Link number and any non-PAD Lane number, then it should send TS1s that match the same Lane number assignments, if possible, or are different if necessary (such as with the optional Lane reversal).

# **Chapter 14: Link Initialization & Training**

---

## **Configuration.Lanenum.Wait**

Prior to discussing the Configuration.Lanenum.Wait state, some background information may be helpful. Lane numbers are assigned sequentially from zero to the maximum number possible for a Link. For example, a x8 Link will be assigned Lane numbers 0 - 7. Ports are required to support a Link as wide as the number of Lanes they have and as small as one Lane. The Lanes will always start with Lane 0 and must be both sequential and contiguous. For example, if some Lanes on a x8 Port aren't working, it might optionally be designed to configure a x4 Link and, if so, it would need to use Lanes 0-3. As another example, if Lane 2 of a x8 Port is not working, it wouldn't be possible to use Lanes 0, 1, 3, and 4 to form a x4 Link because the Lanes wouldn't be contiguous. Any left-over Lanes must send TS1s with Link and Lane set to PAD.

A common timing consideration is repeated many times in the spec for the Configuration substates. Rather than repeat it for every case here, just be aware that it applies in general to both Upstream and Downstream Ports:

To avoid configuring a Link smaller than necessary, it's recommended that a multi-Lane Port delay the final link width evaluation if it sees an error or loses Block Alignment on some Lanes. For 8b/10b, it should wait at least two more TS1s, while for 128b/130b mode it should wait for at least 34 TS1s, but never more than 1ms in any case. The idea is that the Lanes might need settling time after powering up or being reset.

### *Exit to "Detect State"*

After a 2ms timeout if no Link can be configured (e.g.: Lane 0 is not working and Lane Reversal isn't available), or if all Lanes receive two consecutive TS1s with PAD in both the Link and Lane numbers, the link must exit to the Detect State.

## **Downstream Lanes**

### *During Configuration.Lanenum.Wait*

The Downstream Port will continue to transmit TS1s with the non-PAD Link and Lane numbers until one of the exit conditions is met.

### *Exit to "Configuration.Lanenum.Accept"*

If either of the cases listed below is true:

- If two consecutive TS1s have been received on all Lanes with Link and Lane numbers that match what is being transmitted on those Lanes.

# PCI Express Technology

---

- If any Lanes that detected a Receiver see two consecutive TS1s with a Lane number different from when the Lane first entered this substate and at least some Lanes see a non-PAD Link number. The spec points out that this allows the two Ports to settle on a mutually acceptable Link width.

*Exit to “Detect State”*

After a 2ms timeout or if all Lanes receive two consecutive TS1s with Link and Lane numbers set to PAD.

Upstream Lanes

*During Configuration.Lanenum.Wait*

The Upstream Port will continue to transmit TS1s with the non-PAD Link and Lane numbers until one of the exit conditions is met.

*Exit to “Configuration.Lanenum.Accept”*

If either of the cases listed below is true:

- If any Lanes receive two consecutive TS2s.
- If any Lanes receive two consecutive TS1s with a Lane number different from when the Lane first entered this substate and at least some Lanes see a non-PAD Link number.

Note that Upstream Lanes are allowed to wait up to 1ms before changing to that substate, so as to prevent received errors or skew between Lanes from affecting the final Link configuration.

*Exit to “Detect State”*

After a 2ms timeout or if all Lanes receive two consecutive TS1s with Link and Lane numbers set to PAD.

## Configuration.Lanenum.Accept

Downstream Lanes

*During Configuration.Lanenum.Accept*

The Downstream Port has now received TS1s with non-PAD Link and Lane numbers. It is at this point that the Downstream Port must decide if a Link can be established with the Lane numbers returned by the Upstream Port. The three possible state transitions are listed below.

*Exit to “Configuration.Complete”*

If two consecutive TS1s are received with the same non-PAD Link and Lane numbers, and they match the Link and Lane numbers being transmitted in the TS1s for all the Lanes, then Upstream Port has agreed with the Link and

## **Chapter 14: Link Initialization & Training**

---

Lane numbers advertised by the Downstream Port and the next substate is Configuration.Complete. Or if the Lane numbers in the received TS1s are reversed from what the Downstream Port advertised, if the Downstream Port supports Lane Reversal, it can still proceed to Configuration.Complete while using the reversed Lane numbers.

The spec points out that the Reversed Lane condition is strictly defined as Lane 0 receiving TS1s with the highest Lane number (total number of Lanes - 1) and the highest Lane number receiving TS1s with Lane number of zero. One thing that can be understood from this is the answer to a question that comes up in class sometimes: Can the Lane numbers be mixed up, rather than sequential? The answer is no, they must be from 0 to n-1 or from n-1 to 0; no other options are supported.

If the Configuration state was entered from the Recovery state, a bandwidth change may have been requested. If so, status bits will be updated to report the nature of what happened. Basically, the system needs to report whether this change was initiated because the Link wasn't working reliably or because hardware is simply managing the Link power. The bits are updated as follows:

- If the bandwidth change was initiated by the Downstream Port because of a reliability problem, the Link Bandwidth Management Status bit is set to 1b.
- If the bandwidth change was not initiated by the Downstream Port but the Autonomous Change bit in two consecutive received TS1s is cleared to 0b, the Link Bandwidth Management Status bit is set to 1b.
- Otherwise the Link Autonomous Bandwidth Status bit is set to 1b.

### *Exit to "Configuration.Lanenum.Wait"*

If a configured Link can be formed with some but not all of the Lanes that receive two consecutive TS1s with the same non-PAD Link and Lane numbers, those Lanes send TS1s with the same Link number and new Lane numbers. The object is to use a smaller group of Lanes to achieve a working Link.

The new Lane numbers must start with zero and increase sequentially to cover the Lanes that will be used. Any Lanes that don't receive TS1s can't be part of the group and will disrupt the Lane numbering. Any leftover Lanes must send TS1s with Link and Lane set to PAD. For example, if 8 Lanes are available, but Lane 2 doesn't see incoming TS1s, then the Link can't consist of a group that would need Lane 2. Consequently, the x8 and x4 options would not be available, and only a x1 or x2 Link is possible.

# PCI Express Technology

---

*Exit to "Detect State"*

If no Link can be configured, or if all Lanes receive two consecutive TS1s with PAD for Link and Lane numbers.

## Upstream Lanes

*During Configuration.Lanenum.Accept*

The Upstream Port has now received either TS2s or TS1s with non-PAD Link and Lane numbers. It is at this point that the Upstream Port must decide if a Link can be established with the Lane numbers sent by the Downstream Port. The three possible state transitions are listed below.

*Exit to "Configuration.Complete"*

If two consecutive TS2s are received with the same non-PAD Link and Lane numbers, and they match the Link and Lane numbers being transmitted in the TS1s for those Lanes, all is well and the next substate will be Configuration.Complete.

*Exit to "Configuration.Lanenum.Wait"*

If a configured Link can be formed with a subset of Lanes that receive two consecutive TS1s with the same non-PAD Link and Lane numbers, those Lanes send TS1s with the same Link number and new Lane numbers. The object is to use a smaller group of Lanes to achieve a working Link. The next substate in this case will be Configuration.Lanenum.Wait.

As was the case for the Downstream Lanes, the new Lane numbers must start with zero and increase sequentially to cover the Lanes that will be used. Any Lanes that don't receive TS1s can't be part of the group and will disrupt the Lane numbering. Any leftover Lanes must send TS1s with Link and Lane set to PAD.

*Exit to "Detect State"*

If no Link can be configured, or if all Lanes receive two consecutive TS1s with PAD for Link and Lane numbers, then the next state will be Detect.

## Configuration.Complete

This is the only substate of the Configuration state where TS2s are exchanged. As discussed before, the purpose of TS2s is a handshake, or confirmation between the two devices on the link that they are ready to proceed to the next state. So this is the final confirmation of the Link and Lane numbers exchanged in the TS1s leading up to this point.

# **Chapter 14: Link Initialization & Training**

---

It should be noted that Devices are allowed to change their supported data rates and upconfigure capability when they enter this substate, but not while in it. This is because Devices record the capabilities of their Link partner from what is advertised in these TS2s, as will be described in this section.

## **Downstream Lanes**

*During Configuration.Complete*

TS2s are sent using the Link and Lane numbers that match the received TS1s. The TS2s can have the Upconfigure Capability bit set if the Port supports a x1 Link using Lane 0 and is able to up-configure the Link.

For 8b/10b encoding, Lane de-skewing must be completed when leaving this substate. Also, scrambling will be disabled if all configured Lanes see two consecutive TS2s with the Disable Scrambling bit set. The Port that sends these must also disable scrambling. Note that scrambling cannot be disabled when in 128b/130b mode because of the necessary contribution it makes to signal integrity.

The Downstream Port is transmitting TS2s and watching for TS2s coming back. For future reference, record the number of FTSs that must be sent when exiting from the L0s state from the N\_FTS field in the incoming TS2s.

*Exit to “Configuration.Idle”*

The next state will be Configuration.Idle when all Lanes sending TS2s receive 8 TS2s with matching Link and Lane numbers (non-PAD), matching rate identifiers, and matching Link Upconfigure Capability bit in all of them. At least 16 TS2s must also be sent after receiving one TS2.

If the device supports rates greater than 2.5 GT/s, it must record the rate identifier received on any configured Lane and this overrides any previously recorded value. The variable used to track speed changes in Recovery, “changed\_speed\_recovery”, is cleared to zero.

The variable “upconfigure\_capable” is set to 1b if the device sends TS2s with Link Upconfigure Capability set to 1b and receives 8 consecutive TS2s with the same bit set. Otherwise it’s cleared to zero.

Any Lanes that aren’t configured as part of the Link are no longer associated with the LTSSM in progress and must either be:

- Associated with a new LTSSM or
- Transitioned to Electrical Idle
  - a) A special case arises if those Lanes had been configured as part of the Link through L0 previously and LinkUp has remained set at 1b

# PCI Express Technology

---

since then. They must remain associated with the same LTSSM if the Link is upconfigure capable. For that case, it's also recommended that those Lanes leave their Receiver terminations on because they'll become part of the Link again if it is upconfigured. If the terminations aren't left on, they must be turned on from when the LTSSM enters the Recovery.RcvrCfg state all the way through Configuration.Complete. Lanes that weren't part of the Link before can't become part of it through this process, though.

- b) For the optional crosslink, Receiver terminations must be between  $Z_{RX-HIGH-IMP-DC-POS}$  and  $Z_{RX-HIGH-IMP-DC-NEG}$ .
- c) If the LTSSM goes back to Detect, these Lanes will once again be associated with it.
- d) No EIOS is needed before Lanes go to Electrical Idle, and the transition doesn't have to happen on Symbol or Ordered Set boundaries.

After a 2ms timeout:

*Exit to "Configuration.Idle"*

Next state is Configuration.Idle if the `idle_to_rlock_transitioned` variable is less than FFh **and** the current data rate is 8.0 GT/s.

In this transition, the “`changed_speed_recovery`” variable is cleared to zero. Also, the “`upconfigure_capable`” variable may be updated, though it’s not required to do so, if at least one Lane saw eight consecutive TS2s with matching Link and Lane numbers (non-PAD). If the transmitted and received Link Upconfigure Capability bits are 1b, set it to 1b, otherwise clear it to zero.

Lanes that aren't part of the configured Link aren't associated with the LTSSM in progress and have the same requirements as the non-timeout case listed above.

*Exit to "Detect State"*

Otherwise, the next state is Detect.

## Upstream Lanes

*During Configuration.Complete*

TS2s are sent using the Link and Lane numbers that match the received TS2s. The TS2s can have the Upconfigure Capability bit set if the Port supports a x1 Link using Lane 0 and is able to up-configure the Link.

## **Chapter 14: Link Initialization & Training**

---

For 8b/10b encoding, Lane de-skewing must be completed when leaving this substate. Also, scrambling will be disabled if all configured Lanes see two consecutive TS2s with the Disable Scrambling bit set. The Port that sends these must also disable scrambling. Note that scrambling cannot be disabled when in 128b/130b mode because of the necessary contribution it makes to signal integrity.

In this substate, the Upstream Port is receiving TS2s from the Downstream Port, and for future reference, should record the N\_FTS field value number of FTSs that must be sent when exiting from the L0s state from the incoming TS2s.

### *Exit to "Configuration.Idle"*

The next state will be Configuration.Idle when all Lanes sending TS2s receive 8 TS2s with matching Link and Lane numbers (non-PAD), matching rate identifiers, and a matching Link Upconfigure Capability bit in all of them. At least 16 TS2s must also be sent after receiving one TS2.

If the device supports rates greater than 2.5 GT/s, it must record the rate identifier received on any configured Lane, overriding any previously recorded value. The variable used to track speed changes in Recovery, “changed\_speed\_recovery”, is cleared to zero.

The variable “upconfigure\_capable” is set to 1b if the device sends TS2s with Link Upconfigure Capability set to 1b and receives 8 consecutive TS2s with the same bit set. Otherwise it’s cleared to zero.

Any Lanes that aren’t configured as part of the Link are no longer associated with the LTSSM in progress and must either be:

- Optionally associated with a new crosslink LTSSM (if this feature is supported), or
- Transitioned to Electrical Idle
  - a) A special case arises if those Lanes had been configured as part of the Link through L0 previously and LinkUp has remained set at 1b since then. They must remain associated with the same LTSSM if the Link is upconfigure capable. For that case, it’s also recommended that those Lanes leave their Receiver terminations on because they’ll become part of the Link again if it is upconfigured. If they’re not left on, they must be turned on from when the LTSSM enters the Recovery.RcvrCfg state all the way through Configuration.Complete. Lanes that weren’t part of the Link before can’t become part of it through this process, though.

# PCI Express Technology

---

- b) Receiver terminations must be between  $Z_{RX-HIGH-IMP-DC-POS}$  and  $Z_{RX-HIGH-IMP-DC-NEG}$ .
- c) If the LTSSM goes back to Detect, these Lanes will once again be associated with it.
- d) No EIOS is needed before Lanes go to Electrical Idle, and the transition doesn't have to happen on Symbol or Ordered Set boundaries.

After a 2ms timeout:

*Exit to "Configuration.Idle"*

Next state is Configuration.Idle if the idle\_to\_rlock\_transitioned variable is less than FFh **and** the current data rate is 8.0 GT/s.

In this transition, the “changed\_speed\_recovery” variable is cleared to zero. Also, the “upconfigure\_capable” variable may be updated, though it’s not required to do so, if at least one Lane saw eight consecutive TS2s with matching Link and Lane numbers (non-PAD). If the transmitted and received Link Upconfigure Capability bits are 1b, set it to 1b, otherwise clear it to zero.

Lanes that aren’t part of the configured Link aren’t associated with the LTSSM in progress and have the same requirements as the non-timeout case listed above.

*Exit to "Detect State"*

Otherwise, the next state is Detect.

## Configuration.Idle

*During Configuration.Idle*

In this substate, the transmitter is sending Idle data and waiting for the minimum number of received Idle data so this Link can transition to L0. During this time, the Physical Layer reports to the upper layers that the link is operational (Linkup = 1b).

For 8b/10b encoding, the transmitter is sending Idle data on all configured Lanes. Idle data are just data zeros that get scrambled and encoded.

For 128b/130b encoding, the transmitter sends one SDS Ordered Set on all configured Lanes followed by Idle data Symbols. The first Idle Symbol on Lane 0 is the first Symbol of the Data Stream.

# **Chapter 14: Link Initialization & Training**

---

*Exit to “L0 State”*

If using 8b/10b encoding, the next state is L0 if 8 consecutive Idle data symbol times are received on all configured Lanes, and 16 symbol times of idle data were sent after receiving one Idle Symbol.

If using 128b/130b, the next state is L0 if 8 consecutive Idle data are received on all configured Lanes, 16 Idles were sent after receiving one Idle Symbol, and this state wasn't entered by a timeout from Configuration.Complete.

- Lane-to-Lane de-skew must be completed before Data Stream processing begins.
- The Idle Symbols must be received in Data Blocks.
- If software set the Retrain Link bit in the Link Control register since the last transition to L0 from Recovery or Configuration, the Downstream Port must set the Link Bandwidth Management bit in the Link Status register to 1b to indicate that this change was not hardware initiated (autonomous).
- The “idle\_to\_rlock\_transitioned” variable is cleared to 00h on transition to L0.

After a 2ms timeout:

*Exit to “Detailed Recovery Substates”*

If the idle\_to\_rlock\_transitioned variable is less than FFh, the next state is Recovery (Recovery.RcvrLock). Then:

- a) For 8.0 GT/s, increment idle\_to\_rlock\_transitioned by 1.
- b) For 2.5 or 5.0 GT/s, set idle\_to\_rlock\_transitioned to FFh.
- c) NOTE: This variable counts the number of times the LTSSM has transitioned from this state to the Recovery state because the sequence isn't working. The problem may be that equalization hasn't been properly adjusted or that the selected speed just isn't going to work, and the Recovery state will take steps to address these issues. This variable limits the number of these attempts so as to avoid an endless loop. If the Link still isn't working after doing this 256 times (when the count reaches FFh), go back to Detect and start over, hoping for a better result.

*Exit to “Detect State”*

Otherwise (meaning idle\_to\_rlock = FFh), the next state is Detect.

## L0 State

This is the normal, fully-operational Link state, during which Logical Idle, TLPs and DLLPs are exchanged between Link neighbors. L0 is achieved immediately following the conclusion of the Link Training process. The Physical Layer also notifies the upper layers that the Link is ready for operation, by setting the LinkUp variable. In addition, the idle\_to\_rlock\_transitioned variable is cleared to 00h.

### *Exit to “Recovery State”*

The next state will be Recovery if a change in the Link speed or Link width is indicated, or if the Link partner initiates this by going to Recovery or Electrical Idle. Let’s consider each of these three cases in a little more detail in the following discussion.

---

## Speed Change

Two conditions are described in the spec that will cause an automatic change in speed.

The first is when rates higher than 2.5 GT/s are supported by both partners and the Link is active (Data Link Layer reports DL\_Active), or when one partner requests a speed change in its TS Ordered Sets. For example, a Downstream Port will initiate a speed change if a higher rate was noted and software writes the Retrain Link bit and after setting the Target Link Speed field (see Figure 14-26 on page 569) to a different rate than the current rate.

The second condition is when both partners support 8.0 GT/s and one of them wants to perform Tx Equalization. In both conditions the directed\_speed\_change variable will be set to 1b and the changed\_speed\_recovery bit will be cleared to 0b.

A Port will not attempt a speed change (the directed\_speed\_change variable won’t be set) if a rate higher than 2.5 GT/s has never been seen as advertised by the other Port in the Configuration.Complete or Recovery.RcvrCfg substates.

## Chapter 14: Link Initialization & Training

Figure 14-25: Link Control Register

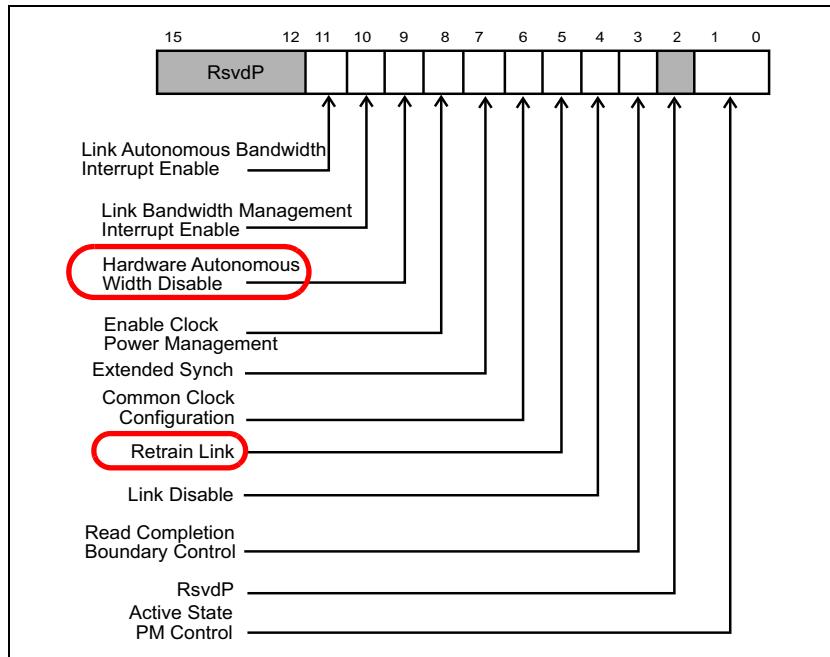
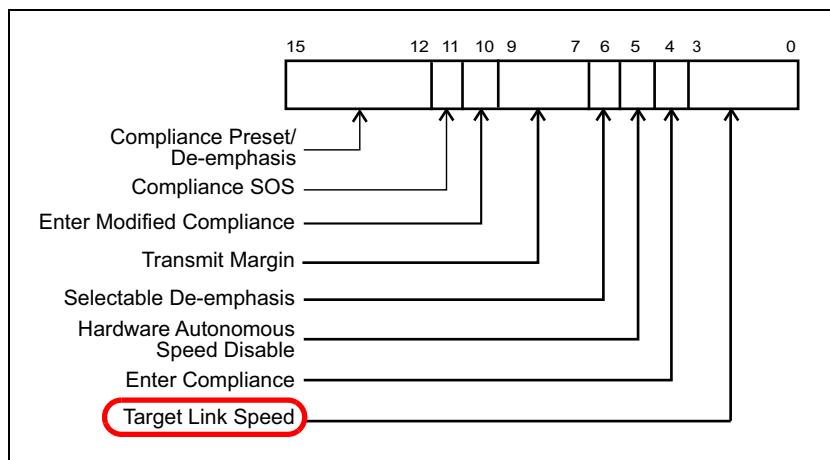


Figure 14-26: Link Control 2 Register



## Link Width Change

An upper layer would normally only direct a Link width reduction if upconfigure\_capable has been set to 1b because otherwise the Link won't be able to go back to the original width. If the Hardware Autonomous Width Disable bit is set to 1b a Port can only reduce the width in an effort to correct a reliability problem. An upper layer can only initiate an increase in Link width if the Link partner advertised that it was upconfigure capable and the Link is not already at its maximum width. Apart from these guidelines, the decision criteria for changing the Link width are not given in the spec and are therefore implementation specific.

---

## Link Partner Initiated

The spec describes three possibilities for this case.

First, if Electrical Idle is detected or inferred (see Table 14-10 on page 596) on all Lanes without first receiving an EIOS on any Lane, the Port may choose to enter Recovery or stay in L0. If errors result from this condition, the Port may be directed to Recovery by means such as setting the Retrain Link bit.

The second case happens when TS1s or TS2s are received (or an EIEOS for 128b/130b) on any configured Lanes, indicating that the Link partner has already entered Recovery. Since both of these cases are initiated by the Link partner, the Transmitter is allowed to complete any TLP or DLLP currently in progress.

Finally, if an EIOS is received on any Lane, indicating a Link power management change, but the Receiver doesn't support L0s and hasn't been directed to L1 or L2, then going to Recovery is the only option.

### *Exit to "L0s State"*

The next state will be L0s for a Transmitter that's been instructed to initiate it, or for a Receiver that sees an EIOS. Interestingly, the LTSSM states for the Transmitter and Receiver of the Port can be different now, because one can be in L0s while the other is still in L0.

- Transmitters go to L0s when directed, if they implement L0s, and send EIOS to initiate the change.
- Receivers go to L0s when an EIOS is seen on any Lane. However, if the Receiver doesn't implement L0s and hasn't been directed to L1 or L2, this will be seen as a problem and the next state will be "Recovery State" instead.

# **Chapter 14: Link Initialization & Training**

---

## *Exit to “Rx\_L0s.Entry”*

The next state will be L1 when one Link partner is directed to initiate this and sends one EIOS on all Lanes (two EIOSs if the speed is 5.0 GT/s) and receives an EIOS on any Lane. Note that both Link partners must have already agreed to enter L1 beforehand and that a Data Link Layer handshake is needed to ensure that both are ready. For more detail on how this works, see the section called “Introduction to Link Power Management” on page 733.

## *Exit to “L2 State”*

The next state will be L2 when one Link partner is directed to initiate this and sends one EIOS on all Lanes (two EIOSs if the speed is 5.0 GT/s) and receives an EIOS on any Lane. Note that both Link partners must have already agreed to enter L2 beforehand and that a handshake is needed to ensure that both are ready. For more detail on how this works, see the section called “Introduction to Link Power Management” on page 733.

---

## **Recovery State**

If everything works as expected, the Link trains to the L0 state without ever going into the Recovery state. But we’ve already discussed two reasons why it might not. First, if the correct Symbol pattern isn’t seen in Configuration.Idle, the LTSSM goes to Recovery in an effort to correct signaling problems by, for example, adjusting equalization values. Secondly, once L0 is reached with a data rate of 2.5 GT/s and both devices support higher speeds, the LTSSM goes to Recovery and attempts to change the Link speed to the highest commonly-supported/advertised speed. In this state, Bit Lock and either Symbol Lock or Block Alignment is re-acquired and the Link is de-skewed again. The Link and Lane Numbers should remain unchanged unless the Link width is being changed. In that case, the LTSSM passes through the Configuration state where Link width is re-negotiated.

NOTE: To simplify the discussion and avoid repeating the same text many times, the term “Lock” will be used here to mean the combination of Bit Lock and either Symbol Lock for 8b/10b encoding or Block Alignment for 128b/130b encoding. A Receiver must acquire this Lock to be able to recognize Symbols, Ordered Sets and Packets.

---

## Reasons for Entering Recovery State

- Exiting the L1 state; Required because there is no fast training option (like sending FTS ordered sets) when exiting L1
- Exiting L0s if the receiver fails to achieve Lock from the FTS ordered sets in the required time, the Link must transition to Recovery
- From L0 if:
  - A higher data rate is available when initial training completes.
  - A Link speed or width change has been requested (for power management or because the current speed or width is unreliable).
  - Software sets the Retrain Link bit in the Link Control Register (see Figure 14-71 on page 644) in an effort to clear transmission problems.
  - An error condition such as a Replay Num Roll-over event associated with the Ack/Nak protocol of the Data Link Layer automatically causes the Physical Layer logic to retrain the Link.
  - Receiver sees TS1s or TS2s on any configured Lane, meaning that the neighbor must have entered Recovery.
  - Receiver sees Electrical Idle on all configured Lanes but did not first receive the Electrical Idle Ordered Set.

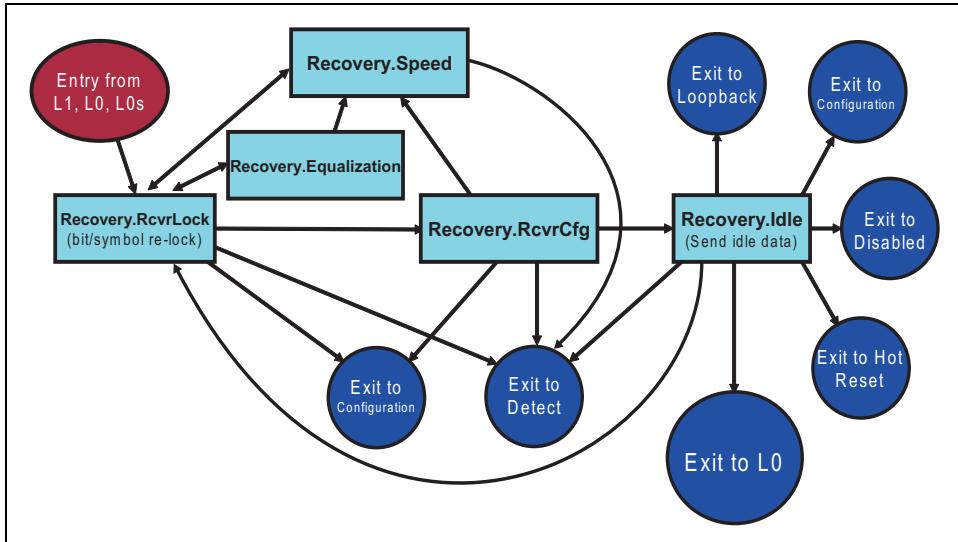
---

## Initiating the Recovery Process

Either Port can initiate Recovery by sending TS1s to its neighbor. When a Port sees incoming TS1s it knows that the other Port has entered Recovery, so it also goes into Recovery and returns TS1s. Both receivers first use the TS1s to reacquire Lock (if necessary) and then proceed to the other substates as needed. This is shown in Figure 14-27 on page 573. A detailed description of what happens in the substates is provided in the sections that follow.

# Chapter 14: Link Initialization & Training

Figure 14-27: Recovery State Machine



## Detailed Recovery Substates

### During Recovery.RcvrLock

Regardless of the speed, Transmitters send TS1s on all configured Lanes using the same Link and Lane numbers that were set in the Configuration state. If the purpose of entering the Recovery state was to change speeds, the speed\_change bit in the Data Rate Identifier Symbol will be set to 1b in the TS1s from the initiating device and the internal variable directed\_speed\_change is set to 1b. This same variable will be set in the other device if the speed\_change bit is set in the incoming TS1s. In addition, The successful\_speed\_negotiation variable is cleared to 0b on entry to this substate.

In this substate, an Upstream Port is allowed to specify the de-emphasis level the Downstream Port should use when operating at 5GT/s. This is accomplished by setting the Selectable De-emphasis bit in its TS1s to the desired value. It's possible that bit errors on the Link will prevent this information from reaching the Downstream Port, so the Upstream Port is allowed to request the de-emphasis level again when going to the Recovery state for a speed change. If the Downstream Port plans to use the requested level, it must record the value of the Selectable De-emphasis bit while in this state.

A new transmitter voltage can also be applied upon entry to this state. The Transmit Margin field in the Link Control 2 register is sampled on entry to this substate and remains in effect until a new value is sampled on another entry to this substate from L0, L0s, or L1.

A Downstream Port that wants to change the rate to 8.0 GT/s and redo the equalization must send EQ TS1s with the speed\_change bit set and advertising the 8.0 GT/s rate. If an Upstream Port receives 8 consecutive EQ TS1s or EQ TS2s with the speed\_change bit set to 1b and the 8.0 GT/s rate supported, it is expected to advertise the 8.0 GT/s rate, too, unless it has concluded that there are reliability problems at that rate that can't be fixed with equalization. Note that a Port is allowed to change its advertised data rates when entering this state, but only those rates that can be supported reliably. And apart from the conditions described here, a device is not allowed to change its supported data rates in this substate or in Recovery.RcvrCfg or Recovery.Equalization.

#### *Exit to "Recovery.RcvrCfg"*

The next state will be Recovery.RcvrCfg if 8 consecutive TS1s or TS2s are received whose Link and Lane numbers match what is being sent *and* their speed\_change bit is equal to the directed\_speed\_change variable *and* their EC field is 00b (if the current data rate is 8.0 GT/s).

- If the Extended Synch bit is set, a minimum of 1024 TS1s in a row must be sent before going to Recovery.RcvrCfg.
- If this substate was entered from Recovery.Equalization, the Upstream Port must compare the equalization coefficients or preset received by all Lanes against the final set of coefficients or preset that was accepted in Phase 2 of the equalization process. If they don't match, it sets the Request Equalization bit in the TS2s it sends.

#### *Exit to "Recovery.Equalization"*

When the data rate is 8.0 GT/s, the Lanes must establish the proper equalization parameters to obtain good signal integrity. This section does not apply for lower speeds. Just because the Link is running at 8.0 GT/s, it does not go through the Recovery.Equalization substate every time Recovery is entered. Recovery.Equalization is only entered if one of these conditions is met:

- If the start\_equalization\_w\_preset variable is set to 1b then:
  - a) Upstream Port registered preset values from the 8 consecutive TS2s it saw prior to changing to 8.0 GT/s. It must use the Transmitter presets and it may optionally use the Receiver presets it received.

## **Chapter 14: Link Initialization & Training**

---

- b) Downstream Port must use the Transmitter presets defined in its Lane Equalization Control register as soon as it changes to 8.0 GT/s and it may optionally use the Receiver presets found there.
- Else (the variable is not set), Transmitters must use the coefficient settings they agreed to when the equalization process was last executed.
  - a) Upstream Port's next state will be Recovery.Equalization if 8 consecutive incoming TS1s have Link and Lane numbers that match those being sent and the speed\_change bit is 0b, but the EC bits are non-zero, indicating that the Downstream Port wishes to redo some parts of the equalization process. The spec notes that a Downstream Port could do this under software or implementation-specific direction. As always, the time it takes to do this must not be allowed to cause transaction timeout errors, which really means the Downstream Port would need to ensure there were no transactions in flight before taking this step.
  - a) Downstream Port's next state will be Recovery.Equalization if directed, as long as this state wasn't entered from Configuration.Idle or Recovery.Idle. The spec points out that no more than two TS1s whose EC=00b should be sent before sending TS1s with a non-zero EC value to request that equalization be redone.

Otherwise, after a 24ms timeout:

*Exit to "Recovery.RcvrCfg"*

The next state will be Recovery.RcvrCfg if both:

- 8 consecutive TS1s or TS2s are received whose Link and Lane numbers match what it being sent and their speed\_change bit is equal to 1b.
- And either the current data rate is already higher than 2.5 GT/s, or at least a higher rate is shown to be supported in the TS1s or TS2s.

*Exit to "Recovery.Speed"*

The next state will be Recovery.Speed if other of the two following conditions are met:

- If the current speed is set higher than 2.5 GT/s but isn't working since entering Recovery (indicated by clearing the variable changed\_speed\_recovery to 0b). The new rate after leaving Recovery.Speed will drop back to 2.5 GT/s.
- If the changed\_speed\_recovery variable is set to 1b, indicating that a higher rate than 2.5 GT/s is already working but the Link was unable to operate at a new negotiated rate. As a result, the operating speed will revert to what it was when Recovery was entered from L0 or L1.

# PCI Express Technology

---

## *Exit to “Configuration State”*

Otherwise, the LTSSM will return to Configuration if a speed change is not requested (directed\_speed\_change variable = 0b and the speed\_change bit in the TS1s and TS2s is 0b), or if the highest commonly supported data rate is 2.5 GT/s.

## *Exit to “Detect State”*

Finally, if none of the other conditions are true, the next state will be Detect.

---

## Speed Change Example

The spec includes an example of a speed change in the discussion of this substate. The scenario is two Link neighbors (device A and device B) that are coming out of reset, both of which support the 5.0 GT/s and 8.0 GT/s rates.

To begin with, the Link will automatically train to L0 using the Gen1 rate of 2.5 GT/s. (This behavior is likely to continue in future spec versions because it provides backward compatibility with older designs.)

In our example both devices support higher rates and this is indicated by the Rate Identifier field in their TS Ordered Sets during training. Both devices note that the other supports a higher rate and one of them (device A) will be the first to set its directed\_speed\_change variable to 1b. When that happens, it will go to Recovery.RcvrLock and send TS1s with the speed\_change bit set. If the desired rate will be 8.0 GT/s and hasn't been before, the devices will exchange EQ TS1s to deliver the TX equalizer presets to be used instead of sending ordinary TS1s.

Device B sees incoming TS1s and also transitions to Recovery.RcvrLock. When it recognizes 8 TS1s in a row with the speed\_change bit set, it responds by setting the speed\_change bit in its own TS1s and goes to Recovery.Speed. Device A waits for that response and, when 8 TS1s in a row with the speed\_change bit have been seen, it goes to Recovery.RcvrCfg and then to Recovery.Speed. In that substate, the transmitters are put into Electrical Idle, the speed is changed to the highest commonly-supported rate, and the directed\_speed\_change variable is cleared.

After a timeout period, both devices transition back to Recovery.RcvrLock and the transmitters are re-activated using the new speed (8.0 GT/s in this case). They send TS1s again now, this time with the speed\_change bit cleared to 0b. If the new speed works well, they transition to Recovery.RcvrCfg and back to L0. However, if device B has a problem, such as failure to achieve Bit Lock, it will timeout in this substate and go back to Recovery.Speed. Device A may have

# **Chapter 14: Link Initialization & Training**

---

already transitioned to Recovery.RcvrCfg by this time, but when it sees Electrical.Idle now, indicating the neighbor has returned to Recovery.Speed, it will also go back to that state. Returning to Recovery.Speed causes both devices to revert to the speed in use when Recovery was entered, 2.5 GT/s in this case, and return to Recovery.RcvrLock.

In response to that development, Device A might set directed\_speed\_change again and try the process a second time. If it failed again, device A might choose to remove the 8.0 GT/s rate from its advertised list and try the speed change again without it. Since the highest common rate is now 5.0 GT/s, if this attempt succeeds the rate will end up at 5.0 GT/s. If it doesn't work, Device A might give up trying to use a higher rate. How and when a device chooses to change its advertised rates or give up trying to get a higher rate working is not given in the spec and will be implementation specific.

---

## **Link Equalization Overview**

This section provides an overview of the Equalization Process and prepares the reader to understand the detailed substate machine behaviors if they are of interest.

Using a higher Link speed results in more signal distortion than lower data rates. To compensate for this and minimize the effort and cost for system designers, the 3.0 spec adds a requirement for Transmitter Equalization. Unlike the fixed de-emphasis values for the lower rates, which is really a simple form of Transmitter equalization itself, the new method uses an active handshake process to match the Transmitters to the actual signaling environment. During this process, each Receiver Lane evaluates the quality of the incoming signal and suggests Tx equalization parameters that the Link partner should use to meet the signal quality requirements.

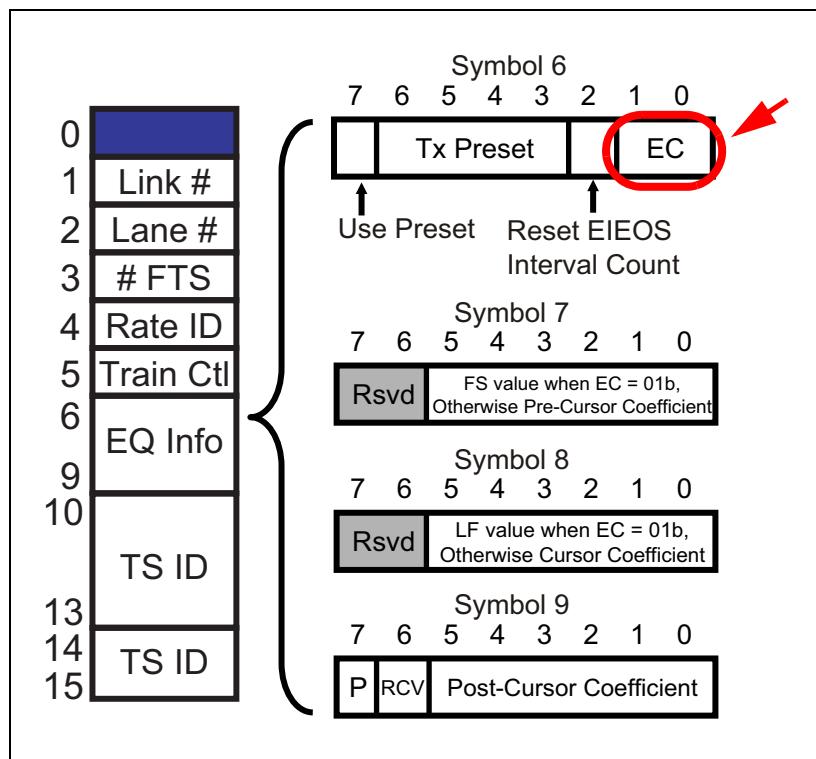
The Link Equalization procedure executes after the first change to the 8.0 GT/s data rate. The spec strongly recommends that the equalization process be initiated autonomously (automatically in hardware) but doesn't require it. If a component chooses not to use the autonomous mechanism then a software-based mechanism must be used. If either port is unable to achieve the necessary signal quality through this process, the LTSSM will conclude that the rate is not working and will go back to Recovery.Speed to request a lower speed.

The process involves up to four phases, as described in the text that follows. Once the speed has been changed to 8.0 GT/s, the current equalization phase in use is indicated by the EC (Equalization Control) field in the TS1s being, as shown in Figure 14-28.

# PCI Express Technology

---

Figure 14-28: EC Field in TS1s and TS2s for 8.0 GT/s



## Phase 0

When the Downstream Port is ready to change from a lower rate to the 8.0 GT/s rate, it enters the Recovery.RcvrCfg sub-state and sends Tx Presets and Rx Hints to the Upstream Port using EQ TS2s as described in “TS1 and TS2 Ordered Sets” on page 510. (Note that this phase is skipped if the Link is already running at 8.0 GT/s.) The Downstream Port (DSP) sends Tx Preset values based on the contents of its Equalization Control register shown in Figure 14-29 on page 579. One thing this highlights is that there can be different equalization values for each Lane. The Downstream Port will use the DSP values for its own Transmitter and optionally for its Receiver, and send the USP values to the Upstream Port for it to use when going to the higher speed.

## Chapter 14: Link Initialization & Training

Figure 14-29: Equalization Control Registers

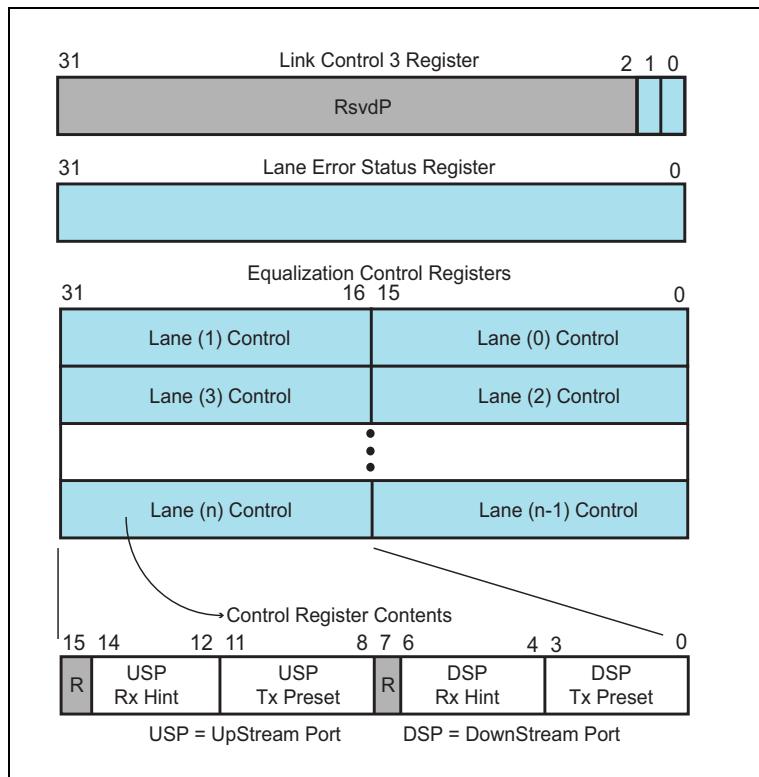


Table 14-8: Tx Preset Encodings

Encoding	De-emphasis	Preshoot
0000b	-6	0
0001b	-3.5	0
0010b	-4.5	0
0011b	-2.5	0
0100	0	0

# PCI Express Technology

---

Table 14-8: Tx Preset Encodings (Continued)

Encoding	De-emphasis	Preshoot
0101	0	2
0110	0	2.5
0111	-6	3.5
1000	-3.5	3.5
1001	0	3.5
1010	Depends on FS and LS values	Depends on FS and LS values
1011b to 1111b	Reserved	Reserved

Table 14-9: Rx Preset Hint Encodings

Encoding	Rx Preset Hint
000b	-6 dB
001b	-7 dB
010b	-8 dB
011b	-9 dB
100	-10 dB
101	-11 dB
110	-12 dB
111	Reserved

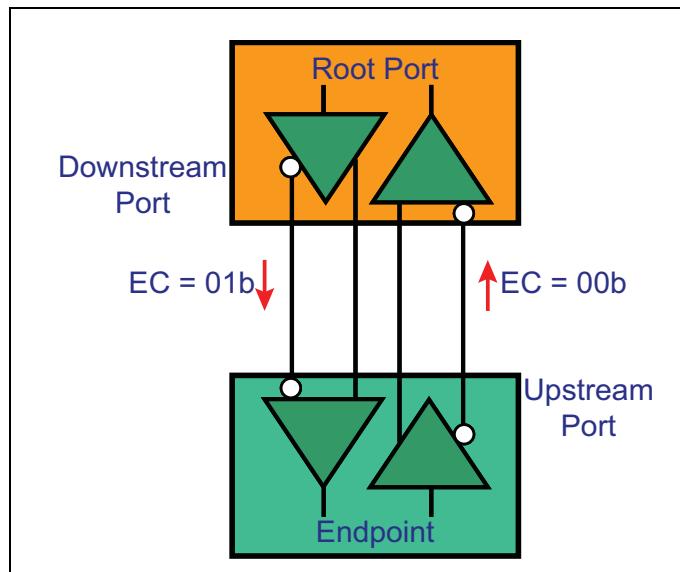
Once the rate does change, the Downstream Port begins in Phase 1 and sends TS1s with EC = 01b. It then waits for the Upstream Port to respond with the same EC value.

Meanwhile, the Upstream Port starts in Phase 0, as illustrated in Figure 14-30 on page 581, and sends TS1s that echo the preset values it received earlier from the

## Chapter 14: Link Initialization & Training

EQ TS1s and EQ TS2s. It will use those requested Tx presets if they're supported, and will optionally use the Rx Hints. The USP is allowed to wait 500ns before evaluating the incoming signal but, once it's able to recognize two TS1s in a row it's ready for the next step. This means the signal quality meets the minimum BER of  $10^{-4}$  (e.g., Bit Error Ratio of less than one error in 10,000 bits). Subsequently the USP sets EC=01b in its TS1s thereby moving to Phase 1 and handing control of the next step to the DSP.

Figure 14-30: Equalization Process: Starting Point



### Phase 1

The DSP performs the same actions as the USP and achieves a BER of  $10^{-4}$  by detecting back-to-back TS1s. During this time, the DSP communicates its Tx presets and FS (Full Swing), LF (Low Frequency), and Post-cursor coefficient values as shown in Figure 14-32 on page 584. The spec gives some additional rules that must be satisfied for a set of requested coefficients, which are:

1.  $|C_{-1}| \leq \text{Floor}(FS/4)$ , (Note: Floor means round down to the integer value)
2.  $|C_{-1}| + C_0 + |C_{+1}| = FS$
3.  $C_0 - |C_{-1}| - |C_{+1}| \geq LF$

# PCI Express Technology

---

FS represents the maximum voltage, and LF defines the minimum voltage as LF/FS. These inform the receiver about the number of possible values and allow the coefficients to be communicated as integer values but understood as fractional values.

As an example, assume we're using the coefficients defined for the P7 preset setting. The FS value acts as a reference and can be any number up to 63 but, for ease of calculation, let's say it's given as 30. In the case of P7,  $C_{-1}$  is -0.1, the value communicated to represent  $C_{-1}$  in the TS1s would be 3, since  $3/30 = 0.1$  and always considered negative.  $C_{+1}$  is -0.2, so it would be communicated as 6, since  $6/30 = 0.2$  and always negative.  $C_0$  is 0.7, so that will be sent as 21, since  $21/30 = 0.7$ . Finally, the LF value represents the smallest possible ratio, and for P7 that is 0.4 times the max value. Consequently, LF will be communicated as 12, since  $12/30 = 0.4$ .

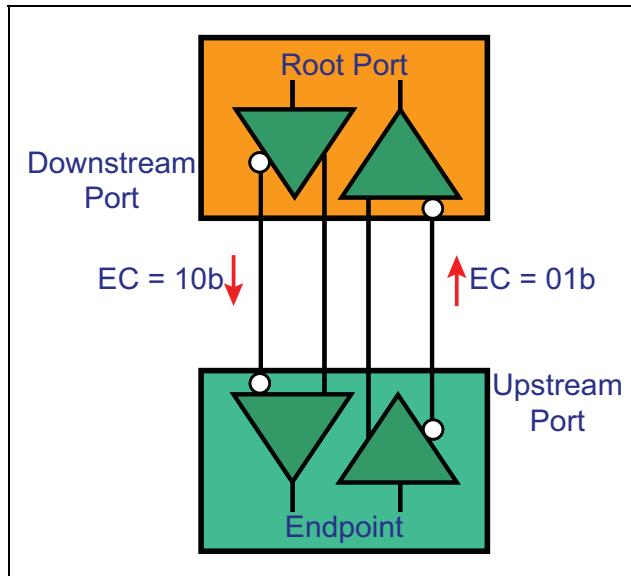
Armed with this information, let's check the three rules to see whether they are satisfied for the P7 case:

1.  $3 \leq \text{Floor}(12/4)$ , This works out to be  $3 \leq 3$  and is true.
2.  $3 + 21 + 6 = 30$  This one is true.
3.  $21 - 3 - 6 \geq 12$  This one is also true, so all three checks are satisfied for P7.

Once the Downstream Port is satisfied that the Link is working well enough to move forward (it recognizes incoming TS1s with EC = 01b), then this phase is complete and it initiates a change to Phase 2 by setting its EC = 10b as illustrated in Figure 14-31 on page 583 and hands control of the next step back to the USP. When the USP responds with EC = 10b, both Ports go to Phase 2. As a happy alternative, the Downstream Port may conclude that the signal quality is already good enough at this point and no further adjustments are necessary. In that case, it set its EC = 00b to exit the equalization process.

# Chapter 14: Link Initialization & Training

Figure 14-31: Equalization Process: Initiating Phase 2



## Phase 2

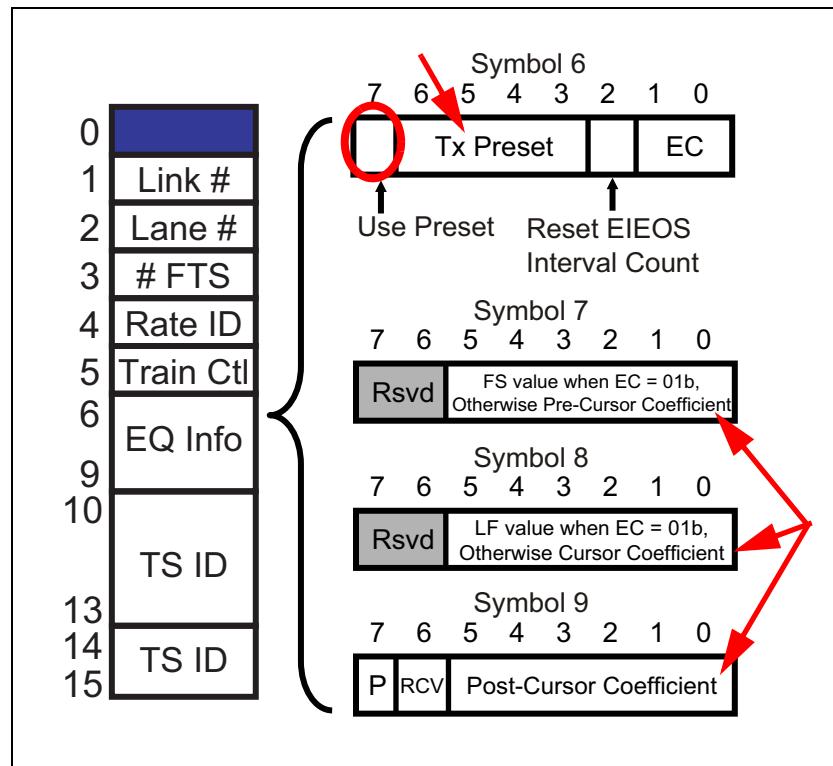
The signal quality has been good enough to recognize TS1s, but not good enough for runtime operation. Once both Ports are in Phase 2, the Upstream Port is allowed to request Tx settings for the Downstream Port and then evaluate how well they work, reiterating the process until it arrives at optimal settings for the current environment. To make a request, it changes the value of the equalization information it sends in its TS1s. As shown in Figure 14-32 on page 584, there are several values of interest:

- **Tx Preset:** The Tx presets are a coarse-grained adjustment to the Transmitter settings that are intended to get it into the right ballpark for the current signaling environment. The Upstream Port sets this value, and sets the “Use Preset” indicator (bit 7 of Symbol 6) to tell the Downstream Port’s Transmitter to use it. If the Use Preset bit is not set, then it’s understood that the presets should stay as they are and that the coefficient values should be changed instead. The Tx coefficients are considered as fine-grained adjustments.

# PCI Express Technology

---

Figure 14-32: Equalization Coefficients Exchanged



- **Coefficients:** Since the spec requires a 3-tap Tx equalizer, three coefficient values are defined that can be pictured as voltage adjustments to a signal pulse that compensates for the distortion it will experience going through the transmission medium, as shown in Figure 14-33 on page 585. This is covered in more detail in the Physical Layer Electrical section titled, “Solution for 8.0 GT/s - Transmitter Equalization” on page 474.
  - **Pre-Cursor Coefficient:** a multiplier applied to the signal prior to the sample point that can boost or reduce the signal depending on the need.
  - **Cursor Coefficient:** the sample point multiplier; always positive.
  - **Post-Cursor Coefficient:** a multiplier applied to the signal after the sample point that can boost or reduce the signal depending on the need.
  - Once the signal meets the quality standard needed, the Upstream Port indicates that it’s ready to move to the next phase by changing EC = 11b.

## Chapter 14: Link Initialization & Training

Figure 14-33: 3-Tap Transmitter Equalization

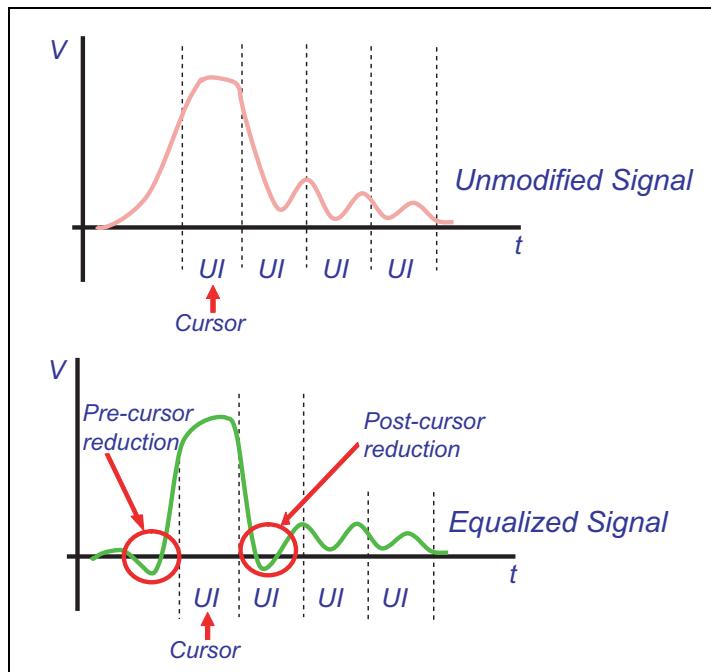
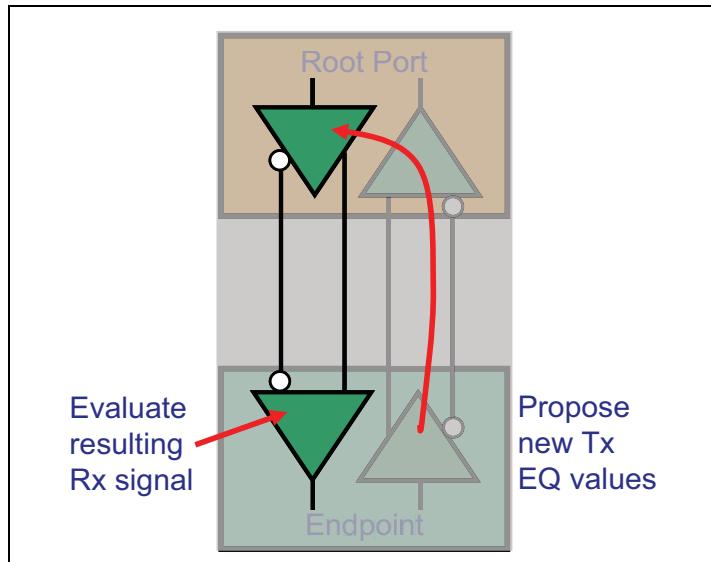


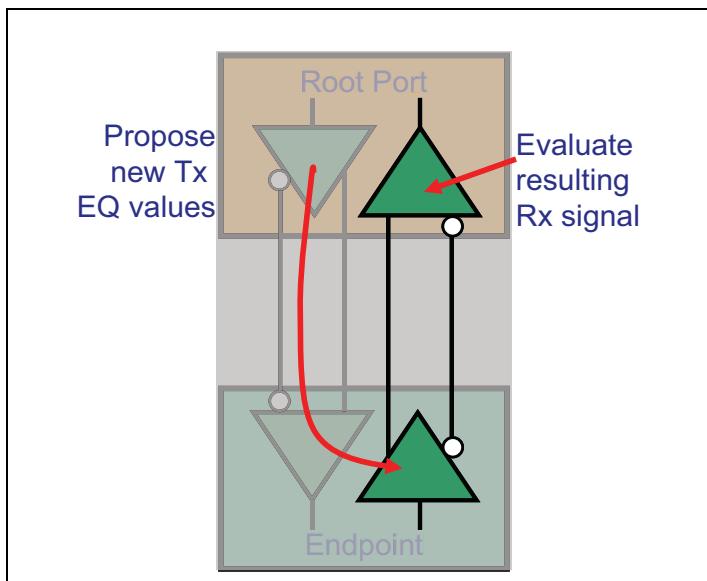
Figure 14-34: Equalization Process: Adjustments During Phase 2



## Phase 3

The Downstream port responds by sending EC = 11b and can now do the same signal evaluation process for the Upstream Port's Transmitter. It sends TS1s that request a new setting the same way: if the Use Preset bit is set, new presets are defined, otherwise new coefficients are being given. This is sent continuously for 1μs or until the request has been evaluated for its result, whichever is later. That evaluation must wait 500ns plus the round trip time through the outgoing logic and back in to the receive logic. Different equalization settings can be tested until one is found that achieves the desired signal quality. At that point the Downstream Port exits the equalization process by setting EC = 00b.

Figure 14-35: Equalization Process: Adjustments During Phase 3



## Equalization Notes

The specification mentions other items associated with the equalization process, as described below:

- All Lanes must participate in the process; even those that may only become active later after an upconfigure event.
- The algorithm used by a component to evaluate the incoming signal and determine the equalization values that its Link partner should use is not given in the spec and is implementation specific.

# **Chapter 14: Link Initialization & Training**

---

- Equalization changes can be requested for any number of Lanes and the Lanes can use different values.
- At the end of the fine-tuning steps (Phase 2 for Upstream Ports and Phase 3 for Downstream Ports), each component is responsible for ensuring that the Transmitter settings cause it to meet the spec requirements.
- Components must evaluate requests to adjust their Transmitter settings and act on them. If valid values are given they must use them and reflect those values in the TS1s they send.
- A request to adjust coefficients may be rejected if the values are not compliant with the rules. The requested values will still be reflected in the TS1s sent back but the Reject Coefficient Values bit will be set.
- Components must store the equalization values that they settled on through this process for future use at 8.0 GT/s. The spec is not explicit on this, but the author's opinion is that these values would survive a change in speed to a lower rate and then back to the 8.0 GT/s rate. That makes sense because it could potentially take a long time to repeat the EQ process and the resulting values would be the same, provided the electrical environment hasn't changed.
- Components are allowed to fine-tune their Receivers at any time, as long as it doesn't cause the Link to become unreliable or go to Recovery.

---

## **Detailed Equalization Substates**

This section covers detailed descriptions of the state machine behaviors during Link Equalization.

### **Recovery.Equalization**

This substate is used to execute the Link Equalization Procedure for 8.0 GT/s and higher rates. The lower rates don't use equalization and the LTSSM won't enter this substate when they're in effect. Since this is a new and complex topic for PCIe, a description of the overall equalization procedure from a high-level view is presented after the state machine details in the section called "Link Equalization Overview" on page 577. First though, let's step through the substates to see the mechanics of the process.

### **Downstream Lanes**

The Downstream Port starts in Phase 1 of the equalization process. To begin this process, there are several bits that need to be reset. In the Link Status 2 register (Figure 14-36 on page 588), the following bits are cleared when entering this substate:

# PCI Express Technology

---

- Equalization Phase 1 Successful
- Equalization Phase 2 Successful
- Equalization Phase 3 Successful
- Link Equalization Request
- Equalization Complete

The Perform Equalization bit of the Link Control 3 register is also cleared to 0b as is the internal variable start\_equalization\_w\_preset. The equalization\_done\_8GT\_data\_rate variable is set to 1b.

Figure 14-36: Link Status 2 Register

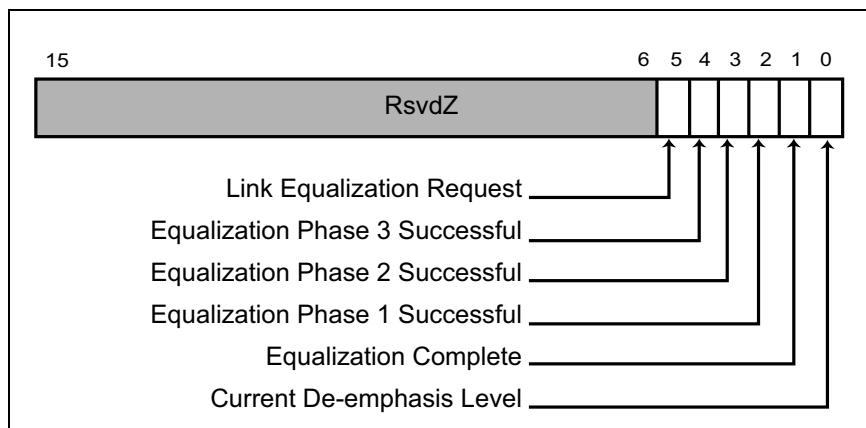
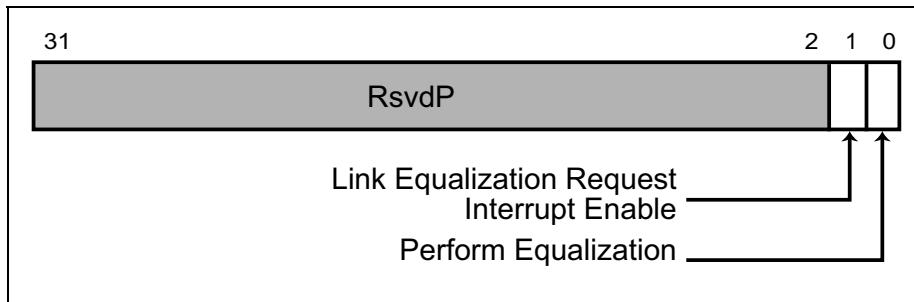


Figure 14-37: Link Control 3 Register



## **Chapter 14: Link Initialization & Training**

---

**Phase 1 Downstream.** During this phase, the Downstream Port sends TS1s with EC = 01b while using the Preset values from the Lane Equalization Control register and with the FS, LF, and Post-cursor Coefficient fields that correspond to the Tx Preset field. It's allowed to wait 500ns before evaluating incoming TS1s if it needs time to stabilize its Receiver logic.

*Exit to "Phase 2 Downstream"*

The Downstream Port will transition to Phase 2 if it wants to continue with the equalization process and when all configured Lanes receive two consecutive TS1s with EC = 01b. At this point, the Port will set the Equalization Phase 1 Successful status bit to 1b and store the received TS1 LF and FS values for use in Phase 3 (if the Downstream Port plans to adjust the Upstream Port's Tx coefficients).

*Exit to "Detailed Recovery Substates"*

If the Downstream Port doesn't want to use Phases 2 and 3, it sets the status bits to 1b (Eq. Phase 1 Successful, Eq. Phase 2 Successful, Eq. Phase 3 Successful, and Eq. Complete). One reason to do this would be because it can already see that the signal characteristics are good enough and the rest of the phases aren't needed.

*Exit to "Recovery.Speed"*

If the consecutive TS1s are not seen after a 24ms timeout, the next state is Recovery.Speed. The successful\_speed\_negotiation flag is cleared to 0b, and the Equalization Complete status bit is set to 1b.

**Phase 2 Downstream.** During this phase, the Downstream Port sends TS1s with EC = 10b and coefficient settings independently assigned on each Lane according to the following:

- If two consecutive TS1s are received with EC = 10b (Upstream Port has entered Phase 2) either for the first time, or with different preset or coefficient values than the last time, and if the values requested are legal and supported, then change the Tx settings to use them within 500ns of the end of the second TS1 requesting them. Also, reflect the values in the TS1s being sent back to the Upstream Port and clear the Reject Coefficient Values bit to 0b. Note that the change must not cause illegal voltages or parameters at the Transmitter for more than 1ns.
  - a) If the requested preset or coefficients are illegal or not supported, don't change the Tx settings but reflect the received values in the

# PCI Express Technology

TS1s being sent and set the Reject Coefficient Values bit to 1b (see Figure 14-38 on page 590).

- If the two consecutive TS1s aren't seen, keep the current Tx preset and coefficient values.

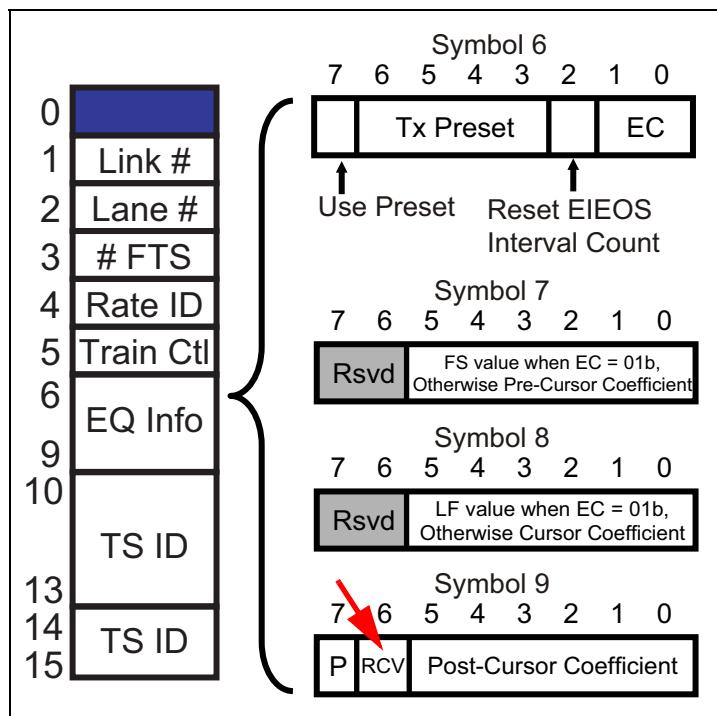
*Exit to "Phase 3 Downstream"*

When the Upstream Port is satisfied with the changes, it begins to send TS1s with EC = 11b, indicating a desire to change to Phase 3. When two consecutive TS1s like this are received, set the Eq. Phase 2 Successful status bit to 1b and change to Phase 3.

*Exit to "Recovery.Speed"*

If after 32 ms, the transition to Phase 3 has not happened, the Port should clear the successful\_speed\_negotiation flag, set the Equalization Complete status bit and exit to the Recovery.Speed substate.

Figure 14-38: TS1s - Rejecting Coefficient Values



## **Chapter 14: Link Initialization & Training**

---

**Phase 3 Downstream.** During this phase, the Downstream Port sends TS1s with EC = 11b and begins the process of evaluating Upstream Tx settings independently for each Lane.

In the transmitted TS1s, the Downstream Port can either request a new preset by setting the Use Preset bit to 1b and Tx Preset field to the desired value, or it can request new coefficients by clearing the Use Preset bit to 0b and setting the Pre-cursor, Cursor, and Post-Cursor Coefficient fields to the desired values. Either request must be made continuously for at least 1μs or until the evaluation has completed. If new preset or coefficient settings are going to be presented, they must be sent on all Lanes at the same time. However, a given Lane isn't required to request new settings if it wants to keep the ones it has.

The Downstream Port must wait long enough to ensure the Upstream Transmitter has had a chance to implement the requested changes, (500ns plus the round-trip delay for the logic), then obtain Block Alignment and evaluate the incoming TS1s. It's not expected that anything useful will be coming from the Upstream Port during the waiting period, and it may not even be legal. That's why obtaining Block Alignment after that time is a requirement.

If two consecutive TS1s are seen that match the same preset or coefficient values that are being requested and don't have the Reject Coefficient Values bit set, then the requested setting was accepted and can be evaluated. If the values match but the Reject Coefficient Values bit is set to 1b, then the requested values have been rejected by the Upstream Port and are not being used. For this case, the spec recommends that the Downstream Port try again with different values but it's not required to do so and may choose to simply exit this phase.

The total time spent on a preset or coefficient request, from the time the request is sent until the completion of its evaluation must be less than 2ms. An exception is available for designs that need more time for the final stage of optimization, but the total time in this phase cannot exceed 24ms and the exception can only be taken twice. If the Receiver doesn't recognize any incoming TS1s, it may assume that the requested setting doesn't work for that Lane.

*Exit to "Detailed Recovery Substates"*

The next state will be Recovery.RcvrLock when all configured Lanes have their optimal settings. When that happens, the Equalization Phase 3 Successful and Equalization Complete status bits will be set to 1b.

## *Exit to “Recovery.Speed”*

Otherwise, after a 24ms timeout (with a tolerance of -0 or +2ms), the next state will be Recovery.Speed, and the successful\_speed\_negotiation flag is cleared to 0b while the Equalization Complete status bit is set to 1b.

## Upstream Lanes

The Upstream Port starts in Phase 0 of the equalization process and must reset several internal bits. In the Link Status 2 register (Figure 14-36 on page 588), the following bits are cleared when entering this substate:

- Equalization Phase 1 Successful
- Equalization Phase 2 Successful
- Equalization Phase 3 Successful
- Link Equalization Request
- Equalization Complete

The Perform Equalization bit of the Link Control 3 register is also cleared to 0b as is the internal variable start\_equalization\_w\_preset. The equalization\_done\_8GT\_data\_rate variable is set to 1b.

**Phase 0 Upstream.** During this phase, the Upstream Port sends TS1s with EC = 00b while using the Tx Preset values that were delivered in the EQ TS2s before entering this state. The equalization information fields in the TS1s being sent must show the preset value and also the Pre-cursor, Cursor, and Post-cursor coefficient fields that correspond to that preset. Note that if a Lane received a reserved or unsupported Tx Preset value in the EQ TS2s, or no EQ TS2s at all, then the Tx Preset field and coefficient values are chosen by a device-specific method for that Lane.

## *Exit to “Phase 1 Upstream”*

When all configured Lanes receive two consecutive TS1s with EC = 01b, indicating that they can recognize the TS1s from the Downstream Port which always starts with this value, then the next phase is Phase 1.

The equalization values LF and FS that are received in the TS1s must be stored and used during Phase 2 if the Upstream Port plans to adjust the Downstream Port’s Tx coefficients.

Upstream Port may wait 500ns after entering Phase 0 before evaluating the incoming TS1s to give time for its Receiver logic to stabilize.

# **Chapter 14: Link Initialization & Training**

---

*Exit to “Recovery.Speed”*

If incoming TS1s are not recognized within a 12ms timeout, the LTSSM will transition to Recovery.Speed, clear the successful\_speed\_negotiation flag and set the Equalization Complete status bit.

**Phase 1 Upstream.** During this phase, the Upstream Port send TS1s with EC = 01b while using the Transmitter settings that were determined in Phase 0. These TS1s contain the FS, LF, and Post-cursor Coefficient values with what is currently being used.

*Exit to “Phase 2 Upstream”*

If all configured Lanes receive two consecutive TS1s with EC = 10b, indicating that the Downstream Port wants to go to Phase 2, then the next phase will be Phase 2, and this Port will set the Equalization Phase 1 Successful status bit.

*Exit to “Detailed Recovery Substates”*

If all configured Lanes receive two consecutive TS1s with EC = 00b, it means that the Downstream Port has decided that the equalization process is already complete and it wants to skip the remaining phases. In this case, the next state will be Recovery.RcvrLock, and the Equalization Phase 1 Successful and Equalization Complete status bits are set to 1b.

*Exit to “Recovery.Speed”*

Otherwise, after a 12ms timeout, the LTSSM will transition to Recovery.Speed, clear the successful\_speed\_negotiation flag and set the Equalization Complete status bit.

**Phase 2 Upstream.** During this phase, the Upstream Port sends TS1s with EC = 10b and begins the process of finding optimal Tx values for the Downstream Port. Recall that the settings are independently determined for each Lane. The process is as follows:

In the transmitted TS1s, the Upstream Port can either request a new preset by putting a legal value in the Transmitter Preset field of the TS1s being sent and setting the Use Preset bit to 1b to tell the Downstream Port to begin using it. Or, request new coefficients by putting legal values in those fields and clearing the Use Preset bit to 0b so the Downstream Port will load them instead of the preset field. Once the request is made it must be repeated for

# PCI Express Technology

---

at least 1 $\mu$ s or until the evaluation is complete. If new preset or coefficient settings are going to be presented, they must be sent on all Lanes at the same time. However, a given Lane isn't required to request new settings if it wants to keep the ones it has.

The Upstream Port must wait long enough to ensure the Downstream Transmitter has had a chance to implement the requested changes, (500ns plus the round-trip delay for the logic), then obtain Block Alignment and evaluate the incoming TS1s. It's not expected that anything useful will be coming from the Downstream Port during the waiting period, and it may not even be legal. That's why obtaining Block Alignment after that time is a requirement.

When TS1s are received that contain the same equalization fields as are being sent and the Reject Coefficient Values bit is not set (0b), then the setting has been accepted and can now be evaluated. If the equalization fields match but the Reject Coefficient Values bit is set (1b), then the setting has been rejected. In that case the spec recommends that the Upstream Port request a different equalization setting, but this is not required.

The total time spent on a preset or coefficient request, from the time the request is sent until the completion of its evaluation must be less than 2ms. An exception is available for designs that need more time for the final stage of optimization, but the total time in this phase cannot exceed 24ms and the exception can only be taken twice. If the Receiver doesn't recognize any incoming TS1s, it may assume that the requested setting doesn't work for that Lane.

*Exit to "Phase 3 Upstream"*

The next phase is Phase 3 if all configured Lanes have their optimal settings. When that happens, the Equalization Phase 2 Successful status bit will be set to 1b.

*Exit to "Recovery.Speed"*

Otherwise, after a 24ms timeout (with a tolerance of -0 or +2ms), the next state will be Recovery.Speed, and the successful\_speed\_negotiation flag is cleared to 0b while the Equalization Complete status bit is set to 1b.

**Phase 3 Upstream.** During this phase, the Upstream Port sends TS1s with EC = 11b and responds to the requested Tx values from the Downstream Port.

## **Chapter 14: Link Initialization & Training**

---

If two consecutive TS1s aren't seen, keep the current Tx preset and coefficient values. However, if two consecutive TS1s are received with EC = 11b (Downstream Port has entered Phase 3) either for the first time, or with different preset or coefficient values than the last time, and if the values requested are legal and supported, then change the Tx settings to use them within 500ns of the end of the second TS1 requesting them. The requested values must be reflected in the TS1s being sent back to the Upstream Port and clear the Reject Coefficient Values bit to 0b. Note that the change must not cause illegal voltages or parameters at the Transmitter for more than 1ns.

- If the requested preset or coefficients are illegal or not supported, don't change the Tx settings but reflect the received values in the TS1s being sent and set the Reject Coefficient Values bit to 1b (see Figure 14-38 on page 590).

*Exit to "Detailed Recovery Substates"*

When the Downstream Port is satisfied with the changes, it begins to send TS1s with EC = 00b, indicating a desire to finish the equalization process. When two consecutive TS1s like this are received, set the Equalization Phase 3 Successful and Equalization Complete status bits to 1b.

*Exit to "Recovery.Speed"*

If the above criteria are not met within a 32 ms timeout, the next state will be Recovery.Speed. The successful\_speed\_negotiation flag will be cleared to 0b and the Equalization Complete status bit will be set.

### **Recovery.Speed**

When entering this substate, a device must enter Electrical Idle on its Transmitter and wait for its Receiver to enter Electrical Idle. After that, it must remain there for at least 800ns if the speed change succeeded (successful\_speed\_negotiation = 1b) or for at least 6 $\mu$ s if the speed change was not successful (successful\_speed\_negotiation = 0b), but not longer than an additional 1ms.

An EIOS must be sent prior to entering this substate if the current rate is 2.5 GT/s or 8.0 GT/s, and two must be sent if the current rate is 5.0 GT/s. An Electrical Idle condition exists on a Lane when these EIOSs have been seen or when it is otherwise detected or inferred (as described in "Electrical Idle" on page 736).

# PCI Express Technology

---

The operating frequency is only allowed to change after the Receiver Lanes have entered Electrical Idle. If the Link is already operating at the highest commonly-supported rate, the rate won't be changed even though this substate is executed.

If the negotiated rate is 5.0 GT/s, the de-emphasis level must be selected based on the setting of the select\_deemphasis variable: if the variable is 0b, apply -6 dB de-emphasis, but if the variable is 1b, apply -3.5 dB de-emphasis instead.

Curiously, the DC common-mode voltage does not have to be maintained within spec limits during this substate.

If this substate is entered after a successful speed negotiation (successful\_speed\_negotiation = 1b), Electrical Idle can be inferred as shown in Table 14-10 on page 596. The spec points out that this covers the case in which both Link partners have recognized incoming TS1s and TS2s, so their absence can be interpreted as an entry to Electrical Idle.

If this substate is entered after an unsuccessful speed negotiation (successful\_speed\_negotiation = 0b), Electrical Idle can be inferred if an Electrical Idle exit has not been detected at least once on any configured Lane in the specified time. This is intended to cover the case when at least one side of the Link is not able to recognize TS Ordered Sets, and so the lack of an exit from Electrical Idle over a longer interval can be treated as an entry to Electrical Idle.

*Table 14-10: Conditions for Inferring Electrical Idle*

State	2.5 GT/s	5.0 GT/s	8.0 GT/s
L0	Absence of Flow Control Update DLLP or SOS in a 128μs window	Absence of Flow Control Update DLLP or SOS in a 128μs window	Absence of Flow Control Update DLLP or SOS in a 128μs window
Recovery.RcvrCfg	Absence of a TS1 or TS2 in a 1280 UI interval	Absence of a TS1 or TS2 in a 1280 UI interval	Absence of a TS1 or TS2 in a 4ms window
Recovery.Speed when successful_speed_negotiation = 1b	Absence of a TS1 or TS2 in a 1280 UI interval	Absence of a TS1 or TS2 in a 1280 UI interval	Absence of a TS1 or TS2 in a 4680 interval

# Chapter 14: Link Initialization & Training

---

*Table 14-10: Conditions for Inferring Electrical Idle (Continued)*

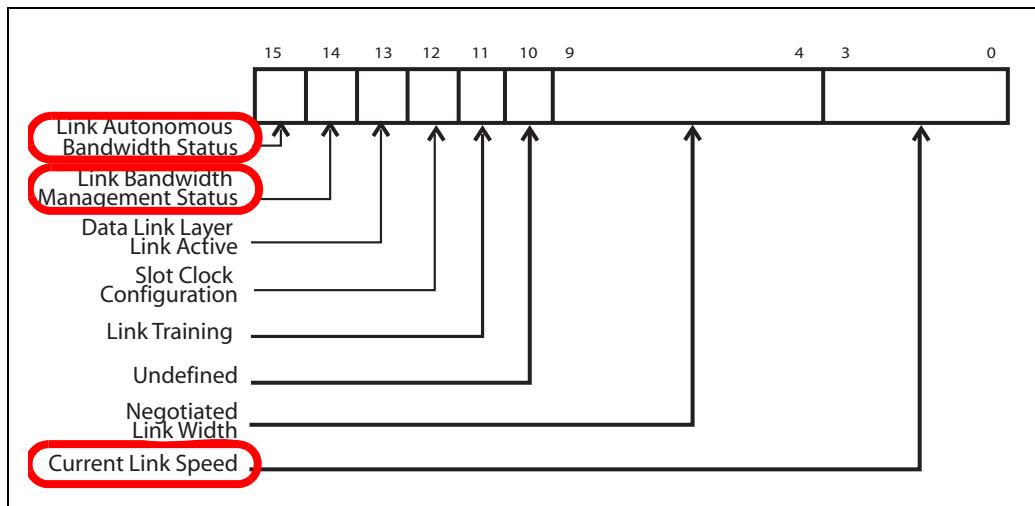
State	2.5 GT/s	5.0 GT/s	8.0 GT/s
Recovery.Speed when successful_speed_neg otiation = 0b	Absence of an Electrical Idle exit in a 2000 UI interval	Absence of an Electrical Idle exit in a 16000 UI interval	Absence of an Electrical Idle exit in a 16000 UI interval
Loopback.Active (as a slave)	Absence of an Electrical Idle exit in a 128μs window	N/A	N/A

The directed\_speed\_change variable will be cleared to 0b and the new data rate must be visible in the Current Link Speed field of the Link Status register, shown in Figure 14-39.

If the speed was changed because of a Link bandwidth change:

- If successful\_speed\_negotiation is set to 1b and the Autonomous Change bit in the 8 consecutive TS2s is set to 1b, or the speed change was initiated by the Downstream Port for autonomous reasons (not a reliability problem and not caused by software setting the Link Retrain bit), then the Link Autonomous Bandwidth Status bit in the Link Status register is set to 1b.
- Otherwise, the Link Bandwidth Management Status bit is set to 1b.

*Figure 14-39: Link Status Register*



# PCI Express Technology

---

## *Exit to “Detailed Recovery Substates”*

Once the timeout has expired, the next state will be Recovery.RcvrLock

If this substate was entered from Recovery.RcvrCfg and the speed change was successful, the new data rate is changed on all the configured Lanes to the highest commonly-supported rate and the changed\_speed\_recovery variable is set to 1b.

If this substate was entered for a second time since entering Recovery from L0 or L1 (indicated by changed\_speed\_recovery = 1b), the new data rate will be the rate that was in use when the LTSSM entered Recovery, and the changed\_speed\_recovery variable is cleared to 0b.

Otherwise, the new data rate will revert to 2.5 GT/s and the changed\_speed\_recovery variable remains cleared to 0b. The spec notes that this represents the case when the rate in L0 was greater than 2.5 GT/s but one Link partner couldn't operate at that rate and timed out in Recovery.RcvrLock the first time through.

## *Exit to “Detect State”*

If none of the conditions for exiting to Recovery.RcvrLock are met, the next state will be Detect, although the spec points out that this shouldn't be possible under normal conditions. It would mean that the Link neighbors can no longer communicate at all.

## **Recovery.RcvrCfg**

This state can only be entered from Recovery.RcvrLock after receiving at least 8 TS1 or TS2 ordered-sets with the same Link and Lane numbers that had been negotiated previously. This means that bit and symbol or block lock have been established and now the Port must determine if there are any other items that need addressed in the Recovery state. If the purpose of entering Recovery was simply to re-establish bit and symbol lock after leaving a link power management state, then it is likely that TS2s will be exchanged here and progress on to Recovery.Idle. If, however, there was another reason for entering the Recovery state (e.g. speed change or link width change), then that will be determined in this substate and the appropriate state transition will occur.

During this substate, the Transmitter sends TS2s on all configured Lanes with the same Link and Lane Numbers configured earlier. If the directed\_speed\_change variable is set to 1b, then the speed\_change bit in the TS2s must also be set. The N\_FTS value in the TS2s should reflect the number needed at the current rate. The start\_equalization\_w\_preset variable is cleared to 0b when entering this substate.

## **Chapter 14: Link Initialization & Training**

---

If the speed has been changed a different N\_FTS number may now be seen in the TS2s. That value must be used for exiting future L0s low-power Link states. For 8b/10b encoding, Lane-to-Lane de-skew must be completed before leaving this substate. Devices must note the advertised rate identifier in incoming TS2s and use this to override any previously-recorded values. When using 128b/130b encoding, devices must make a note of the value of the Request Equalization bit for future reference.

Notes about this substate: The variable successful\_speed\_negotiation is set to 1b. The data rates advertised in the TS2s with the speed\_change bit set are noted at this point for future reference, as is the Autonomous Change bit for possible logging in the Link Status register during Recovery.Speed. The rate that will be selected in Recovery.Speed will be the highest commonly-supported rate. Interestingly, the change to Recovery.Speed will take place for this case even if the Link is already operating at the highest supported rate, although in that case the rate won't actually change.

If the speed is going to change to 8.0 GT/s, a Downstream Port will need to send EQ TS2s (bit 7 of Symbol 6 is set to 1b to indicate an EQ training sequence). This case would be recognized if 8.0 GT/s is mutually supported and 8 consecutive TS1s or TS2s have been seen on any configured Lane with the speed\_change bit set, or if the equalization\_done\_8GT\_data\_rate variable is 0b, or if directed.

An Upstream Port can set the Request Equalization bit if the current data rate is 8.0 GT/s and there was a problem with the equalization process. Either Port can request equalization be done again by setting both the Request Equalization and Quiesce Guarantee bits to 1b.

Upstream Ports set their select\_deemphasis variable based on the Selectable De-emphasis bit in the received TS2s. And, if the TS2s were EQ TS2s, they set the start\_equalization\_w\_preset variable to 1b and update their Lane Equalization register with the new information (i.e.: update the Upstream Port Transmitter Preset and Receiver Preset Hint fields in the register). Any configured Lanes that don't receive EQ TS2s will choose their preset values for 8.0 GT/s operation in a design-specific manner. Downstream Ports must set their start\_equalization\_w\_preset variable to 1b if the equalization\_done\_8GT\_data\_rate variable is cleared to 0b or if directed.

Finally, if 128b/130b encoding is in use, devices must make a note of the Request Equalization bit. If set, both it and the Quiesce Guarantee bit must be stored for future reference.

# PCI Express Technology

---

*Exit to “Recovery.Idle”*

The next state will be Recovery.Idle if two conditions are true:

- Eight consecutive TS2s are received on any configured Lane with Link and Lane numbers and rate identifiers that match those being sent and either:
  - a) The speed\_change bit in the TS2s is cleared to 0b, or
  - b) No rate higher than 2.5 GT/s is commonly supported.
- Sixteen TS2 have been sent after receiving one and they haven't been interrupted by any intervening EIEOS. The changed\_speed\_recovery and directed\_speed\_change variables are both cleared to 0b on entry to this substate.

*Exit to “Recovery.Speed”*

The LTSSM will go to Recovery.Speed if ALL three conditions listed below are true:

- Eight consecutive TS2s are received on any configured Lane with the speed\_change bit set, identical rate identifiers, identical values in Symbol 6, and:
  - a) The TS2s were standard 8b/10b TS2s, or
  - b) The TS2s were EQ TS2s, or
  - c) 1ms has expired since receiving eight EQ TS2s on any configured Lane.
- Both Link partners support rates higher than 2.5 GT/s, or the rate is already higher than 2.5 GT/s.
- For 8b/10b encoding, at least 32 TS2s were sent with the speed\_change bit set to 1b without any intervening EIEOS after receiving one TS2 with the speed\_change bit set to 1b in the same configured Lane. For 128b/130b encoding, at least 128 TS2s are sent with the speed\_change bit set to 1b after receiving one TS2 with the speed\_change bit set to 1b in the same configured Lane.

A transition to Recovery.Speed can also occur if the rate has changed to a mutually negotiated rate since entering Recovery from L0 or L1 (changed\_speed\_recovery = 1b) and any configured Lanes have either seen EIOS or detected/inferred Electrical Idle and haven't seen TS2s since entering this substate. This means a higher rate was attempted but the Link partner indicates that it isn't working for some reason. The new rate will return to whatever it was when Recovery was entered from L0 or L1.

The final case that can cause a transition to Recovery.Speed is if the rate has *not* changed to a mutually negotiated rate since entering Recovery from L0

## **Chapter 14: Link Initialization & Training**

---

or L1 (changed\_speed\_recovery = 0b), and the current rate is already higher than 2.5 GT/s, and any configured Lanes have either seen EIOS or detected/inferred Electrical Idle and haven't seen TS2s since entering this substate. In this case, the understanding is that the current rate isn't working and the solution is to drop back down, so the new rate will become 2.5 GT/s.

### *Exit to "Configuration State"*

The next state will be Configuration if 8 consecutive TS1s are received on any configured Lane with Link or Lane numbers that don't match those being sent and either the speed\_change bit is cleared to 0b, or no rate higher than 2.5 GT/s is commonly supported.

The variables changed\_speed\_recovery and directed\_speed\_change are cleared to 0b when the LTSSM transitions to Configuration. If the N\_FTS value has changed since last time, the new value must be used for L0s going forward.

### *Exit to "Detect State"*

After 48ms without resolving to one of the previously-defined state transitions, the next state will be Detect if the data rate is 2.5 GT/s or 5.0 GT/s.

If the rate is 8.0 GT/s there is another possibility because the number of attempts may not have been exceeded yet. That is indicated by the idle\_to\_rlock\_transitioned variable, and if it's less than FFh when the rate is 8.0 GT/s, the new state will be "Recovery.Idle". If that transition is made, the variables changed\_speed\_recovery and directed\_speed\_change will be cleared to 0b. However, once idle\_to\_rlock\_transitioned reaches FFh, and the 48ms timeout is seen, the next state will be Detect.

## **Recovery.Idle**

As the name implies, Transmitters will usually send Idles in this substate as a preparation for changing to the fully operational L0 state. For 8b/10b mode, Idle data is normally sent on all the Lanes, while for 128b/130b an SDS is sent to start a Data Stream and then Idle data Symbols are sent on all the Lanes.

### *Exit to "L0 State"*

The next state is L0 if either of the following cases is true. In either case, if the Retrain Link bit has been written to 1b since the last transition to L0 from Recovery or Configuration, the Downstream Port will set the Link Bandwidth Management Status bit to 1b (see Figure 14-39 on page 597).

- 8b/10b encoding is in use and 8 consecutive Symbol Times of Idle data have been received and 16 Idle data Symbols have been sent since the first one was received.

# PCI Express Technology

---

- 128b/130b encoding in use, 8 consecutive Symbol Times of Idle data have been received and 16 Idle data Symbols have been sent since the first one was received, and this state wasn't entered from Recovery.RcvrCfg. Note that Idle data Symbols must be contained in Data Blocks, Lane-to-Lane De-skew must be completed before Data Stream processing starts, and the idle\_to\_rlock\_transitioned variable is cleared to 00h on transition to L0.

*Exit to "Configuration State"*

The next state is Configuration if either:

- A Port is instructed by a higher layer to optionally reconfigure the Link, such as to change the Link width.
- Any configured Lane sees two consecutive incoming TS1s with Lane numbers set to PAD (a Port that transitions to Configuration to change the Link will send PAD Lane numbers on all Lanes). The spec recommends that the LTSSM use this transition when changing the Link width to reduce the time it will take.

*Exit to "Disable State"*

The next state is Disabled if either:

- A Downstream or optional crosslink Port is instructed by a higher layer to set the Disable Link bit in its TS1s or TS2s.
- Any configured Lane of an Upstream or optional crosslink Port sees the Disable Link bit set in two consecutive incoming TS1s.

*Exit to "Hot Reset State"*

The next state is Hot Reset if either:

- A Downstream or optional crosslink Port is instructed by a higher layer to set the Hot Reset bit in its TS1s or TS2s.
- Any configured Lane of an Upstream or optional crosslink Port sees the Hot Reset bit set in two consecutive incoming TS1s.

*Exit to "Loopback State"*

The next state is Loopback if either:

- A Transmitter is known to be Loopback Master capable (design specific; the spec does not provide a means to verify this) and instructed by a higher layer to set the Loopback bit in its TS1s or TS2s.
- Any configured Lane of an Upstream or optional crosslink Port sees the Loopback bit set in two consecutive incoming TS1s. The receiving device then becomes the Loopback slave.

# Chapter 14: Link Initialization & Training

*Exit to “Detect State”*

Otherwise, after a 2ms timeout, the next state will be Detect unless the idle\_to\_rlock\_transitions variable is less than FFh, in which case the next state will be “Detailed Recovery Substates”. For the transition to Recovery.RcvrLock, if the data rate is 8.0 GT/s the idle\_to\_rlock\_transitions variable is incremented by 1b, while for 2.5 or 5.0 GT/s it will be set to FFh.

---

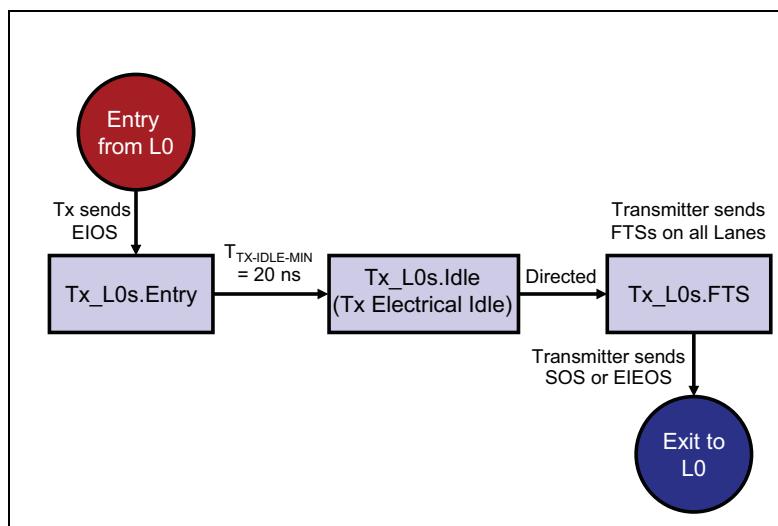
## L0s State

This is the low power Link state that has the shortest exit latency back to L0. Devices manage entry and exit from this state automatically under hardware control without any software involvement. Each direction of a Link, can enter and exit the L0s state independent of each other.

### L0s Transmitter State Machine

The L0s state has different substates for the Transmitter and the Receiver. The Transmitter substates will be described first. As shown in Figure 14-40 on page 603 the transmitter state machine associated with L0s state is a simple one.

Figure 14-40: L0s Tx State Machine



## **Tx\_L0s.Entry.**

A Transmitter enters L0s when directed by an upper layer. The spec gives no decision criteria for this, but intuitively it would occur based on an inactivity timeout: no TLPs or DLLPs being sent for a given time. To enter L0s, the Transmitter sends one EIOS (two EIOSs for the 5.0 GT/s rate) and enters Electrical Idle. The Transmitter is not turned off, however, and must maintain the DC common-mode voltage within the spec range.

*Exit to “Tx\_L0s.Idle”*

The next state will be Tx\_L0s.Idle after the TX-IDLE-MIN timeout (20ns). This time is intended to ensure that the Transmitter has established the Electrical Idle condition.

## **Tx\_L0s.Idle.**

In this substate, the transmitter continues the Electrical Idle state until directed to leave. Because this direction of the Link is in Electrical Idle, there will be a power savings benefit, which is the entire purpose of the L0s state.

*Exit to “Tx\_L0s.FTS”*

The next state will be Tx\_L0s.FTS when directed, such as when the Port needs to resume packet transmission. The LTSSM will be instructed in a design-specific manner to exit this state.

## **Tx\_L0s.FTS.**

In this substate, the Transmitter will start sending FTS ordered sets to retrain the Receiver of the Link Partner. The number of FTSs sent is the N\_FTS value advertised by the Link Partner in its TS Ordered Sets during the last training sequence that led to L0. The spec notes that if a Receiver times out while trying to do this, it may choose to increase the N\_FTS value it advertises during the Recovery state.

If the Extended Synch bit is set (see Figure 14-71 on page 644), the transmitter must send 4096 FTSs instead of the N\_FTS number. This extends the time available to synchronize external test and analysis logic, which may not be able to recover Bit Lock as quickly as the embedded logic can.

For all data rates, no SOSs can be sent prior to sending any FTSs. However, for the 5.0 GT/s rate, 4 to 8 EIE Symbols must be sent prior to sending the FTSs. For 128b/130b, an EIEOS must be sent prior to the FTSs.

# Chapter 14: Link Initialization & Training

*Exit to “L0 State”*

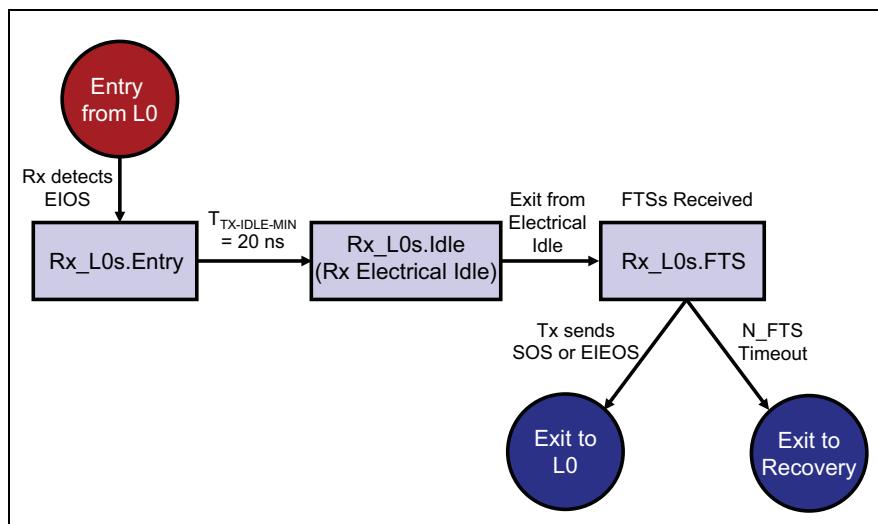
The Transmitter will transition to the L0 state once all the FTSs have been sent and:

- a) For 8b/10b encoding, one SOS is sent on all configured Lanes, although none are sent before or during the FTSs.
- b) For 128b/130b encoding, one EIEOS is sent followed by an SDS and a Data Stream.

## L0s Receiver State Machine

Figure 14-41 on page 605 shows the Receiver L0s state machine. A Receiver is required to implement L0s support if the ASPM Support field in the Link Capability register shows it to be supported, and is allowed to implement it even if that support is not indicated.

*Figure 14-41: L0s Receiver State Machine*



# PCI Express Technology

---

## Rx\_L0s.Entry.

Entered when a Receiver that receives an EIOS, provided it supports L0s and hasn't been directed to L1 or L2.

*Exit to "Rx\_L0s.Idle"*

The next state will be Rx\_L0s.Idle after the T<sub>TX-IDLE-MIN</sub> timeout (20ns).

## Rx\_L0s.Idle.

The Receiver is now in Electrical Idle mode and is just waiting to see an exit from Electrical Idle.

As an aside regarding Electrical Idle, the early versions of the spec expected that Electrical Idle would be based on a squelch-detect circuit measuring a voltage threshold. Later, as speeds increased, detecting such small voltage differences became increasingly difficult. Consequently, more recent spec versions allow Electrical Idle to be inferred by observing Link behavior, rather than actually measuring the voltage. However, if the voltage level isn't used to detect entry into Electrical Idle, then it also can't be used to detect an exit from it. To handle that problem, a new Ordered Set was introduced called the EIEOS (Electrical Idle Exit Ordered Set). The EIEOS consists of alternating bytes of all zeros and all ones and creates the effect of a low-frequency clock on the Lanes. Once a Receiver has entered Electrical Idle it can watch for this pattern on the signal to inform it that the Link is exiting from Electrical Idle.

*Exit to "Rx\_L0s.FTS"*

The next state will be Rx\_L0s.FTS after the Receiver detects an exit from Electrical Idle.

## Rx\_L0s.FTS.

In this substate, the Receiver has noticed an exit from Electrical Idle and is now trying to re-establish Bit and Symbol or Block lock on the incoming bit stream (which are really FTS ordered sets).

*Exit to "L0 State"*

The next state will be L0 if an SOS is received in 8b/10b encoding or an SDS in 128b/130b encoding on all configured Lanes. The Receiver must be able to accept valid data immediately after that, and Lane-to-Lane de-skew must be completed before leaving this state.

# **Chapter 14: Link Initialization & Training**

---

*Exit to “Recovery State”*

Otherwise the next state will be Recovery after the N\_FTS timeout. If so, the Transmitter must also go to Recovery, although it's allowed to finish any TLP or DLPP that was in progress. If the timeout occurs, the spec recommends that the N\_FTS value be increased to reduce the likelihood of it happening again. The N\_FTS timeout is defined as follows:

For 8b/10b, the minimum timeout is given as  $40 * [N\_FTS + 3] * UI$ , while the maximum allowed is twice that time. Since 10 bits (UI represents one bit time) are needed per Symbol, this works out to  $(4 * N\_FTS + 12)$  Symbols. The extra 12 Symbols are explained as 6 for a max-sized SOS + 4 for the possible extra FTS + 2 more for Symbol margin. In summary, then, the minimum time is the time it should take to send the requested number of FTSs plus 12 Symbols, while the maximum time is twice as much as that.

If the extended synch bit is set, the min time = 2048 FTSs and the max time = 4096 FTSs. The actual timeout value a Receiver will use must also take into account the 4 to 8 EIE Symbols for speeds other than 2.5 GT/s.

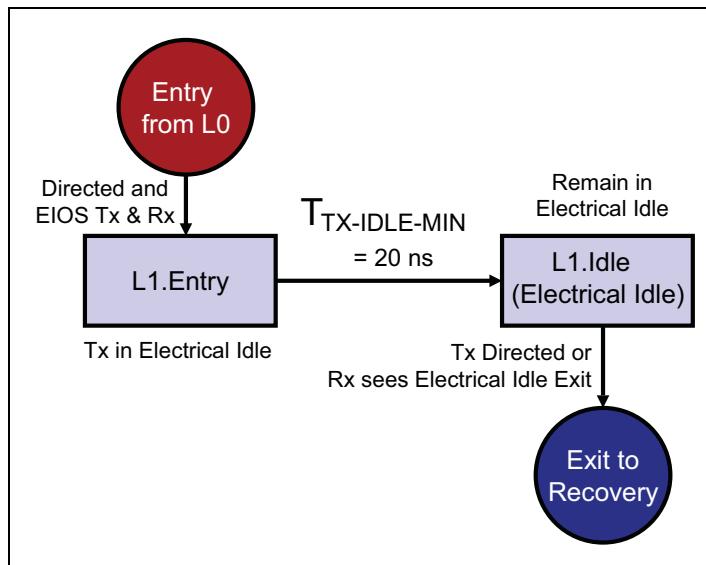
For 128b/130b, the timeout value is given as a minimum of  $130 * [N\_FTS + 5 + 12 + \text{Floor}(N\_FTS/32)] * UI$  and a max of twice that time. The value  $130 * UI$  means 130 bit times which represents one Block, so if we remove those two values we can say we're looking at  $[N\_FTS + 5 + 12 + \text{Floor}(N\_FTS/32)]$  Blocks. The value  $[5 + \text{Floor}(N\_FTS/32)]$  represents the EIEOSs that will need to be sent during this time. One EIEOS will be sent after every 32 FTSs, so  $\text{Floor}(N\_FTS/32)$  gives that number. The other 5 are accounted for by the first EIEOS, the last EIEOS, the SDS, the periodic EIEOS and an additional EIEOS in case the Transmitter chooses to send two EIEOS followed by an SDS when N\_FTS is divisible by 32. Finally, the value of 12 represents the number of SOSs that will be sent if the extended synch bit is set. When that bit is set, the timeout will use  $N\_FTS = 4096$ .

---

## **L1 State**

This Link power state trades a longer exit latency for more aggressive power management compared to the L0s state. L1 is an option for ASPM, like L0s, meaning devices can enter and exit this state automatically under hardware control without any software involvement. However, unlike L0s, software is also able to direct an Upstream Port to initiate a change to L1, and it does so by writing the device power state to a lower level (D1, D2, or D3). The L1 state is also different from L0s in that it affects both directions of the Link.

Figure 14-42: L1 State Machine



Since going to Electrical Idle can indicate a desire by the Link partner to enter L0s, L1 or L2, differentiating which should be the next state is handled by having both partners agree beforehand when they're going to enter L1. A handshake informs them that the partner is ready and it's therefore safe to proceed. For more detail on how this works, see the section called "Introduction to Link Power Management" on page 733. Figure 14-42 on page 608 shows the L1 state machine, which is described in the following sections.

## L1.Entry

In order for an Upstream Port to enter this state, it must send a request to enter L1 to its Link Partner and receive acknowledgement that it is OK to put the Link into L1. (The reason for requesting to go into L1 may be because of ASPM or because of software involvement.) Once the L1 request acknowledge is received, the Upstream Port enters the L1.Entry substate.

In order for a Downstream Port to enter this state, it must receive an L1 enter request from the Upstream Port and send a positive response to that request. Then the Downstream Port waits to receive an Electrical Idle Ordered Set (EIOS) and have its receive lanes drop to Electrical Idle. It is at this point that the Downstream Port enters the L1.Entry substate.

# **Chapter 14: Link Initialization & Training**

---

## *During L1.Entry*

All configured Transmitters send an EIOS and enter Electrical Idle while maintaining the proper DC common mode voltage.

## *Exit to "L1.Idle"*

The next state will be L1.Idle after the  $T_{TX-IDLE-MIN}$  timeout (20ns). This time is intended to ensure that the Transmitter has established the Electrical Idle condition.

## **L1.Idle**

During this substate, Transmitters remain in the Electrical Idle.

For rates other than 2.5 GT/s the LTSSM must remain in this substate for at least 40ns. In the spec, this delay is said “to account for the delay in the logic levels to arm the Electrical Idle detection circuitry in case the Link enters L1 and immediately exits”.

## *Exit to "Recovery State"*

The next state will be Recovery when a Transmitter is directed to change it or when any Receiver detects an exit from Electrical Idle. Reasons for leaving L1 include the need to deliver a DLLP or TLP, or a desire to change the Link width or speed. If a speed change is desired, a Port is allowed to set the `directed_speed_change` variable to 1b and must clear the `changed_speed_recovery` variable to 0b. Optionally, the Port may exit L1 and then initiate the speed change later by setting `directed_speed_change` to 1b and entering Recovery from L0 instead.

---

## **L2 State**

This is a deeper power state with a longer exit latency than L1. Power Management software directs an Upstream Port to initiate entry into L2 (both directions of the Link go to L2) when its device is placed in the D3Cold power state and the appropriate Link handshakes have been completed.

Main power will be shut off by the system once it learns that everything is ready. When power is removed, the Link power state will become either L2 or L3, depending on whether a secondary power source called  $V_{AUX}$  (auxiliary voltage) is available. If  $V_{AUX}$  is present, the Link enters L2; if not, it enters L3.

The motivation for L2 is to use the small power available from  $V_{AUX}$  to inform the system when an event has occurred for which the Link needs to have power

## **PCI Express Technology**

---

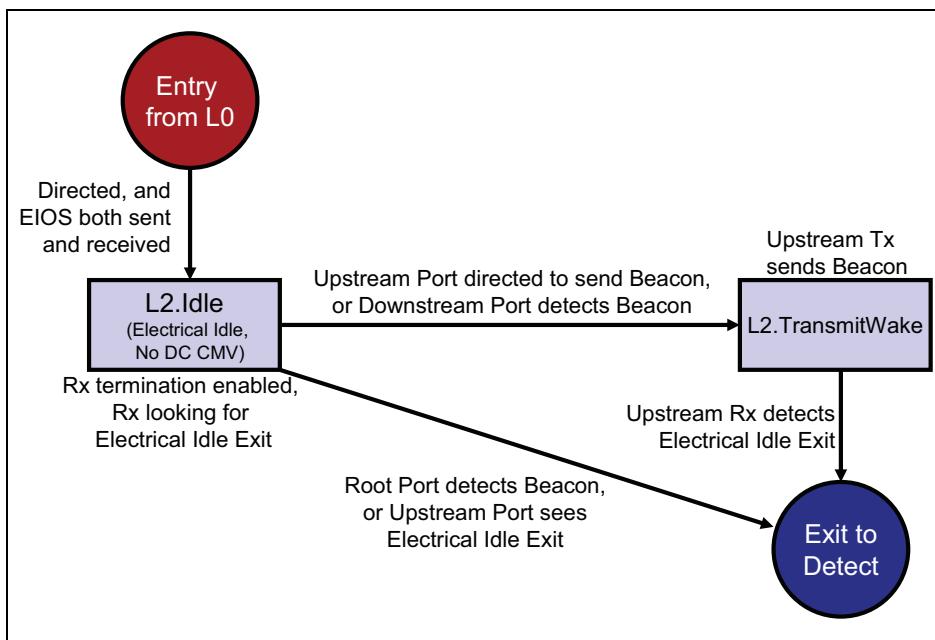
restored. There are two standard ways a device can inform the system of such an event. One is a side-band signal called the WAKE# pin, and the other is an in-band signal called a “Beacon.” The L2 state isn’t needed for WAKE#, but is required if the optional Beacon will be used. The spec explicitly states that devices operating at 5.0 or 8.0 GT/s don’t need to support Beacon, so it would seem that this is legacy support and only interesting for devices operating at 2.5 GT/s. For more detail on Link wakeup options, refer to “Waking Non-Communicating Links” on page 772.

If supported, the Beacon is a low-frequency (30 KHz - 500 MHz) in-band signal that an Upstream Port supporting wakeup capability must be able to send on at least Lane 0 and a Downstream Port must be able to receive. Intermediate devices like Switches that receive a Beacon on a Downstream Port must forward it to their Upstream Port. The ultimate destination for the Beacon is the Root Complex, because that’s where the system power control logic is expected to reside.

A Transmitter going to Electrical Idle could indicate a desire to enter any of the low-power Link states (L0s, L1 or L2), so a means of differentiating them is needed. For L2, this is handled by having the Link partners agree beforehand that they’re going to enter L2 by using a handshake sequence to ensure that they’re both ready. For more detail on how this works, see the section called “Introduction to Link Power Management” on page 733. Figure 14-43 on page 611 shows the L2 entry and Exit state machine, which is described in the following text.

# Chapter 14: Link Initialization & Training

Figure 14-43: L2 State Machine



## L2.Idle

To enter this substate, all the necessary handshake process must have already taken place between both ports on the Link and the ports have sent and received the required EIOS.

All configured Transmitters must remain in the Electrical Idle state for at least the  $T_{TX-IDLE-MIN}$  timeout (20ns). However, since the main power will now be shut off, they aren't required to maintain the DC common-mode voltage within the spec range. Receivers won't start looking for the Electrical exit condition until at least after the 20ns timeout expires. All Receiver terminations must remain enabled in the low impedance condition.

### Exit to "L2.TransmitWake"

The next state will be L2.TransmitWake if the Upstream Port is instructed to send a Beacon (the Beacon is always and only directed upstream to the Root Complex).

# PCI Express Technology

---

If the Downstream Port of a Switch detects a Beacon, it must direct the Upstream Port of the Switch to exit to L2.TranmitWake and begin sending a Beacon.

## *Exit to “Detect State”*

Once main power is returned, the next state will be Detect.

If this Port has main power, but it detects an exit from Electrical Idle on any “predetermined” Lanes, meaning those that could be negotiated to be Lane 0 (multi-Lane Links must have at least two predetermined Lanes), the next state will be detect. When this happens to a Switch Upstream Port, the Switch must also transition its Downstream Ports to Detect.

## **L2.TranmitWake**

During this substate, the Transmitter will send the Beacon on at least Lane 0. Note that this state only applies to Upstream Ports because only they can send a Beacon.

## *Exit to “Detect State”*

The next state will be Detect if an Electrical Idle exit is detected on any Receiver of an Upstream Port. Of course, power must have already been restored to the devices in order for the neighbor to exit from Electrical Idle.

---

## **Hot Reset State**

A Port enters the Hot Reset state either because it is a Bridge and software programmed its configuration space to propagate a Hot Reset Downstream as explained in “Hot Reset (In-band Reset)” on page 837, because a Port received two consecutive TS1s with the Hot Reset bit asserted.

## *During Hot Reset*

A Port transmits TS1s with the Hot Reset bit set continuously but doesn’t change the configured Link and Lane Numbers.

If the Upstream Port of Switch enters the Hot Reset state, all configured Downstream Ports must transition to Hot Reset as soon as possible.

## *Exit to “Detect State”*

In the Bridge where Hot Reset was originated, once software clears the configuration space bit that initiated the Hot Reset, the Bridge Port enters Detect. However, the Port must remain in the Hot Reset state for a minimum of 2ms.

# **Chapter 14: Link Initialization & Training**

---

For Ports where Hot Reset was entered because of receiving two consecutive TS1s with the Hot Reset bit asserted, it remains in this state as long as it continues to receive these type of TS1s. Once the Port stops receiving TS1s with the Hot Reset bit asserted, it will transition to the Detect state. However, the Port must remain in the Hot Reset state for a minimum of 2ms.

---

## **Disable State**

A Disabled Link is Electrically Idle and does not have to maintain the DC common mode voltage. Software initiates this by setting the Link Disable bit (see Figure 14-71 on page 644) in the Link Control register of a device and the device then sends TS1s with the Link Disable bit asserted.

### *During Disable*

All Lanes transmit 16 to 32 TS1s with the Disable Link bit asserted, send an EIOS (two consecutive EIOSs for the 5.0 GT/s case) and then transition to Electrical Idle. The DC common-mode voltage does not need be within spec.

If an EIOS (two consecutive EIOSs for the 5.0 GT/s case) was sent and an EIOS was also received on any configured Lane, then LinkUp = 0b (False) and the Lanes are considered to be disabled.

### *Exit to “Detect State”*

For Upstream Ports, the next state will be Detect when Electrical Idle is detected at the Receiver or if no EIOS has been received within a 2ms time-out.

For Downstream Ports, the next state will also be Detect, but not until the Link Disable bit has been cleared to 0b by software.

---

## **Loopback State**

The Loopback state is a test and debug feature that isn't used during normal operation. A device acting as a Loopback master can put the Link partner into the Loopback slave mode by sending TS1s with the Loopback bit asserted. This can be done in-circuit, allowing the possibility of using the Loopback state to perform a BIST (Built In Self Test) on the Link.

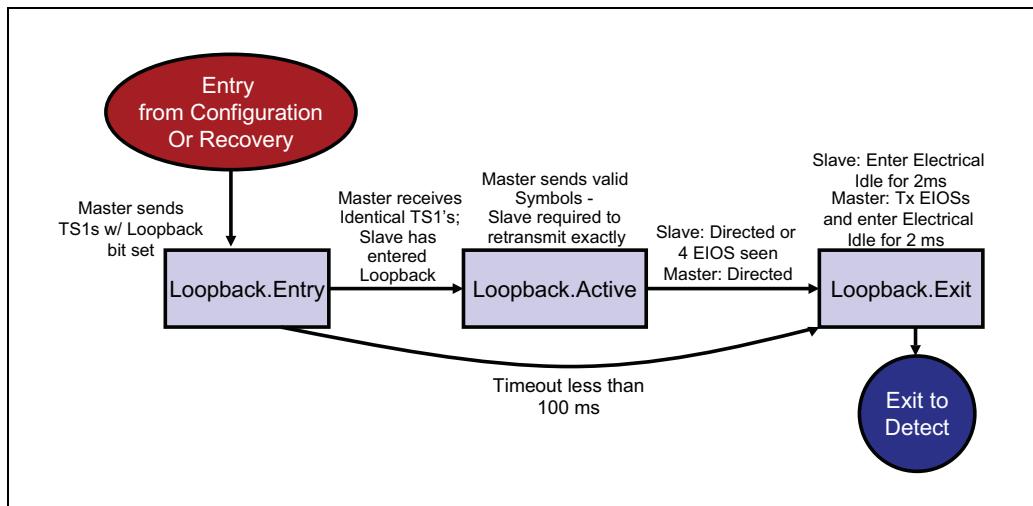
Once in this state, the Loopback master sends valid Symbols to the Loopback slave, which then echoes them back. The Loopback slave continues to perform

# PCI Express Technology

clock tolerance compensation, so the master must continue to insert SOSs at the correct intervals. To perform clock tolerance compensation, the Loopback slave may have to add or delete SKP Symbols to the SOS it echoes to the Loopback master.

The Loopback state is exited when the Loopback master transmits an EIOS and the receiver detects Electrical Idle. The Loopback state machine is shown in Figure 14-44 on page 614 and described in the following text.

Figure 14-44: Loopback State Machine



## Loopback.Entry

The typical behavior for this substate is for the Loopback Master to send TS1s with the Loopback bit set until it starts seeing those TS1s being returned. Once the Loopback Master sees TS1s being returned with the Loopback bit asserted, it knows that its Link Partner is now behaving as the Loopback Slave and is simply repeating everything it receives.

While in this substate, the Link is not considered to be active ( $\text{LinkUp} = 0\text{b}$ ). Also, the Link and Lane numbers used in TS1s and TS2s are ignored by the Receiver. The spec makes an interesting observation regarding the use of Lane numbers with 128b/130b encoding. As it turns out, each Lane uses a different seed value for its scrambler (see “Scrambling” on page 430). Consequently, if the Lane numbers haven’t been negotiated before going into the Loopback mode, it’s possible that the Link partners could have different Lane assignments and would therefore be unable to recognize incoming Symbols. This can be avoided by waiting until the Lane numbers have been negotiated before direct-

# **Chapter 14: Link Initialization & Training**

---

ing the master to go to the Loopback state, or by directing the master to set the Compliance Receive bit during Loopback.Entry, or by some other method.

## **Loopback Master:**

In this substate, the Loopback Master will continuously send TS1s with the Loopback bit set. The master may also assert the Compliance Receive bit in the TS1s to help testing when one or both Ports are having trouble obtaining bit lock, Symbol lock, or Block alignment after a rate change. If the bit is set it must not be cleared while in this state.

If this substate was entered from Configuration.Linkwidth.Start, check to see whether the speed in use is the highest mutually supported rate for both Link partners. If not:

- Change to the highest common speed. Send 16 TS1s with the Loopback bit set followed by an EIOS (two EIOSs if the current speed is 5.0 GT/s), and then go to Electrical Idle for 1ms. During the idle time, change the speed to the highest commonly-supported rate.
- If the highest common rate is 5.0 GT/s, the slave's Tx de-emphasis is controlled by the master setting its Selectable De-emphasis bit in the TS1s to the desired value ( $1b = -3.5$  dB,  $0b = -6$  dB).
- For data rates of 5.0 GT/s and higher, the master's Transmitter can choose any de-emphasis settings it wants, regardless of the settings it sent to the slave.
- Potential problem: if Loopback is entered after the Link has already trained to L0 and LinkUp = 1b, it's possible for one Port to enter Loopback from Recovery and the partner to enter from Configuration. If that happened, the latter Port might try to change the speed while the Port entering from Recovery does not, resulting in a situation where the results are undefined. The spec states that the test set-up must avoid conflicting cases like this.

### *Exit to "Loopback.Active"*

The next state will be Loopback.Active after either 2ms, if the Compliance Receive bit is set in the outgoing TS1s, or two consecutive TS1s are received on a design-specific number of Lanes with the Loopback bit set and the Compliance Receive bit was not set in the outgoing TS1s.

Note that if the speed was changed, the master must ensure that enough TS1s have been sent for the slave to be able to acquire Symbol lock or Block alignment before going to the Loopback.Active state.

## *Exit to "Loopback.Exit"*

If neither of the conditions to enter Loopback.Active are met, the next state will be Loopback.Exit after a design-specific timeout of less than 100ms.

## **Loopback Slave:**

This substate is entered by receiving two consecutive TS1s with the Loopback bit asserted.

If this substate was entered from Configuration.Linkwidth.Start, check to see whether the speed in use is the highest one that mutually supported by both Link partners. If not:

- Change to the highest common speed. Send one EIOS (two EIOSs if the current speed is 5.0 GT/s), and then go to Electrical Idle for 2ms. During the idle time, change the speed to the highest commonly-supported rate.
- If the highest common rate is 5.0 GT/s, set the Transmitter's de-emphasis according to the Selectable De-emphasis bit in the received TS1s (1b = -3.5 dB, 0b = -6 dB).
- If the highest common rate is 8.0 GT/s and:
  - a) EQ TS1s directed the slave to this state, use the Tx Preset settings they specified.
  - b) Normal TS1s directed the slave to this state, the slave is allowed to use its default transmitter settings.

## *Exit to "Loopback.Active"*

The next state will be Loopback.Active if the Compliance Receive bit was set in the incoming TS1s that directed the slave to this state. The slave doesn't need to wait for particular boundaries to send looped-back data, and is allowed to truncate any Ordered Set in progress.

Otherwise, the slave sends TS1s with Link and Lane numbers set to PAD and the next state will be Loopback.Active if:

- The rate is 2.5 or 5.0 GT/s and Symbol lock is acquired on all Lanes.
- The rate is 8.0 GT/s and two consecutive TS1s are seen on all active Lanes. Equalization is handled by evaluating and applying the values given in the TS1s, as long as they're supported and the EC value is appropriate for the direction of the Port (10b for Downstream Ports,

# **Chapter 14: Link Initialization & Training**

---

and 11b for Upstream Ports). Optionally, the Port can accept either of the EC values for this case. If the settings are applied, they must take effect within 500ns of receiving them and must not cause the Transmitter to violate any electrical specs for more than 1ns. A significant difference compared to the process in Recovery.Equalization is that the new settings are not echoed in the TS1s being sent by the slave.

- For 8b/10b, the slave must only transition to looped-back data on a Symbol boundary, but is allowed to truncate any Ordered Set in progress. For 128b/130b, no boundary is specified for when the looped-back data can be sent, and it is still allowed to truncate any Ordered Set in progress.

## **Loopback.Active**

During this substate, the Loopback Master sends valid encoded data and should not send EIOS until it's ready to exit Loopback. The Loopback Slave echoes the received information without modification (even if the encoding is determined to be invalid), with the possible exception of inverting the polarity as determined in the Polling state. The slave also continues to perform clock tolerance compensation. That means SKPs must be added or removed as needed, but the Lanes aren't required to all send the same number.

*Exit to "Loopback.Exit"*

The next state will be Loopback.Exit for the loopback master if directed.

The next state will be Loopback.Exit for the loopback slave if either of two conditions is true:

- The slave is directed to exit or four consecutive EIOSs are seen on any Lane.
- Optionally, if the current speed is 2.5 GT/s and an EIOS is received or Electrical Idle is detected or inferred on any Lane. Electrical Idle may be inferred if any configured Lane has not detected an exit from Electrical Idle for 128 $\mu$ s.

The slave must be able to detect an Electrical Idle on any Lane within 1ms of EIOS being received. Between the time EIOS is received and Electrical Idle is actually detected, the Loopback Slave may receive a bit stream that is undefined by the encoding scheme, and it may loop that back to the transmitter.

## Loopback.Exit

During this substate, the Loopback Master sends an EIOS for Ports that support only 2.5 GT/s and eight consecutive EIOSs for Ports that support rates higher than 2.5 GT/s (optionally send 8 for the Ports that only support 2.5 GT/s, too), and then enter Electrical Idle on all Lanes for 2ms.

- The Loopback Master must transition to Electrical Idle within  $T_{TX-IDLE-SET-TO-IDLE}$  after sending the last EIOS. Note that the EIOS marks the end of the master's transmit and compare operations. Any data received by the master after any EIOS is received is undefined and should be ignored.

The loopback slave must enter Electrical Idle on all Lanes for 2ms but must echo back all Symbols received prior to detecting Electrical Idle to ensure that the master sees the arrival of the EIOS as the end of the logical send and compare operation.

*Exit to “Detect State”*

The next state will be Detect once the required EIOSs have been exchanged and the Lanes have been in Electrical Idle for 2ms.

---

## Dynamic Bandwidth Changes

Higher data rates and wider Links for PCIe offer higher performance than previous generations but use more power, too. Consequently, the 2.0 spec writers chose to include another pair of power management mechanisms that allow the hardware to adjust the Link speed and width on the fly. These allow the Link to use the highest speed and widest possible Link when performance is needed, or to drop down to a lower speed or narrower Link width or both to reduce power. There are two clear advantages to this method compared to changing the Link or Device power state.

First, the Link is always able to communicate regardless of the changes, with a relatively short interruption in service to make the change. Second, the power saving can be greater. For example, a x16 Link would almost certainly use less power operating as an active x1 Link than as a x16 Link in L0s.

Secondly, in addition to power conservation, bandwidth reductions can also be used to resolve reliability problems. For example, it may be that a high speed Link produces unacceptable reliability, in which case either Link component is allowed to remove the offending speed from the list of supported speeds that it advertises. How a component makes that reliability determination is not speci-

## **Chapter 14: Link Initialization & Training**

---

fied. Interestingly, components are also permitted to go into the Recovery state and advertise a different set of supported speeds without requesting a speed change in the process.

Changing the Link Speed or Link Width requires the Link to be re-trained. When the Link is in the L0 state, and the speed needs to be changed, the LTSSM of the port desiring the speed change starts transmitting TS1s to its neighbor. Doing so results in the two involved ports' LTSSMs going through Recovery state where the Link speed is changed and then back to L0.

Similarly, the port that desires to change the Link width starts transmitting TS1s to its neighbor. Doing so results in the two involved ports' LTSSMs going through Recovery state then Configuration state where the Link width is changed. The LTSSM finally returns to L0 with the new Link width established.

Because the LTSSM is involved in dynamic Link bandwidth management, it makes sense to discuss the two aspects of Link bandwidth management, dynamic Link speed change and dynamic Link width change in the following sections. Let's consider these two options separately, starting with Link speed changes.

---

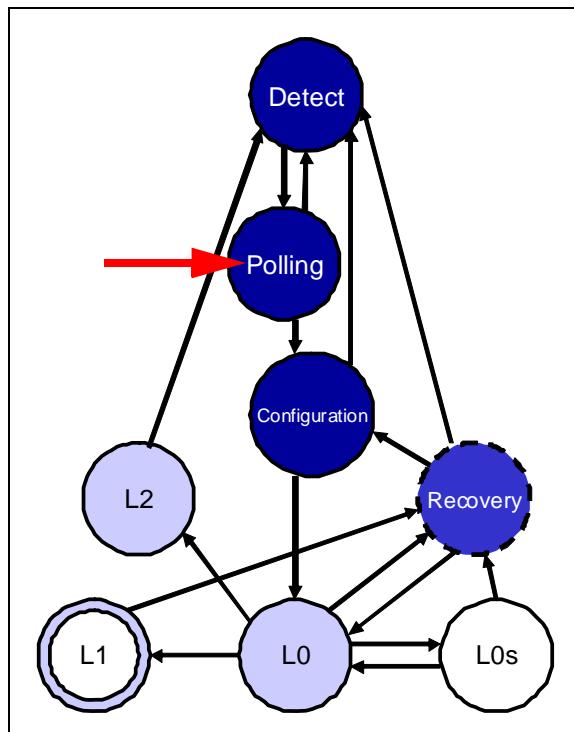
### **Dynamic Link Speed Changes**

By way of review, the LTSSM states are illustrated in Figure 14-45 on page 620 to make it easier to recall the flow of states. Although according to the Gen1 specification, speed change was indicated to be performed in the Polling state, the subsequent Gen2 spec moved this function to the Recovery state.

# PCI Express Technology

---

Figure 14-45: LTSSM Overview



During the Polling state, TS1s are exchanged between Link neighbors, and these contain several kinds of information as shown in Figure 14-46 on page 621. The most interesting part for us here is byte number 4, the Rate Identifier. Bits 1, 2 and 3 indicate which data rates are available and the spec points out that 2.5 GT/s must always be supported, while 5.0 GT/s must also be supported if 8.0 GT/s is supported.

The meaning of bit 6 depends on whether the Port is facing upstream or downstream and also on what LTSSM state the Port is in. However, for the speed change case the options are reduced because it's only meaningful coming from the Upstream Port and just indicates whether or not the speed change is an autonomous event. "Autonomous" means that the Port is requesting this change for its own hardware-specific reasons and not because of a reliability issue. Bit 7 is used by the Upstream Port to request a speed change. These values are very similar in the TS2s, although bit 6 has another meaning now related to autonomous Link width changes that we'll discuss later.

## Chapter 14: Link Initialization & Training

Figure 14-46: TS1 Contents

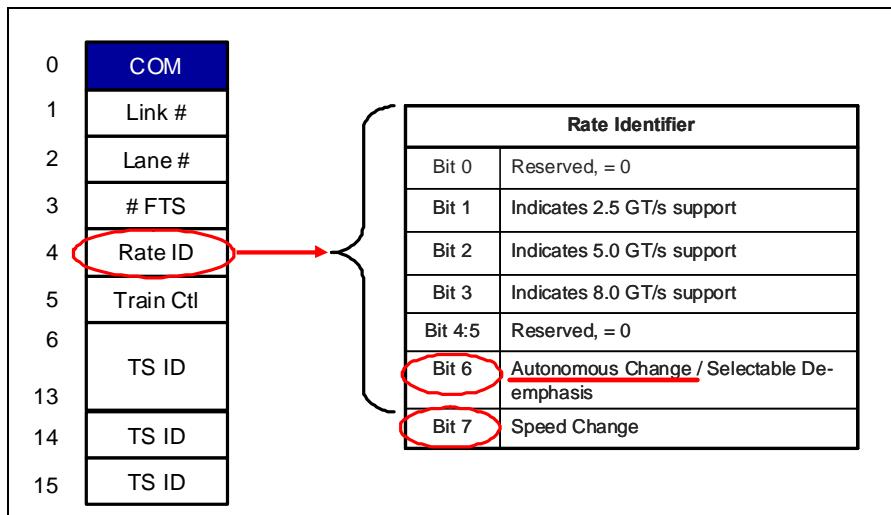
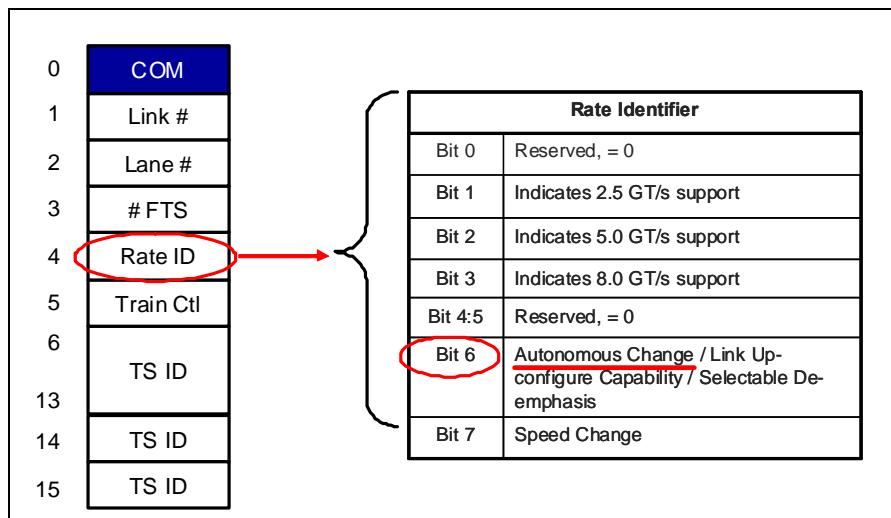


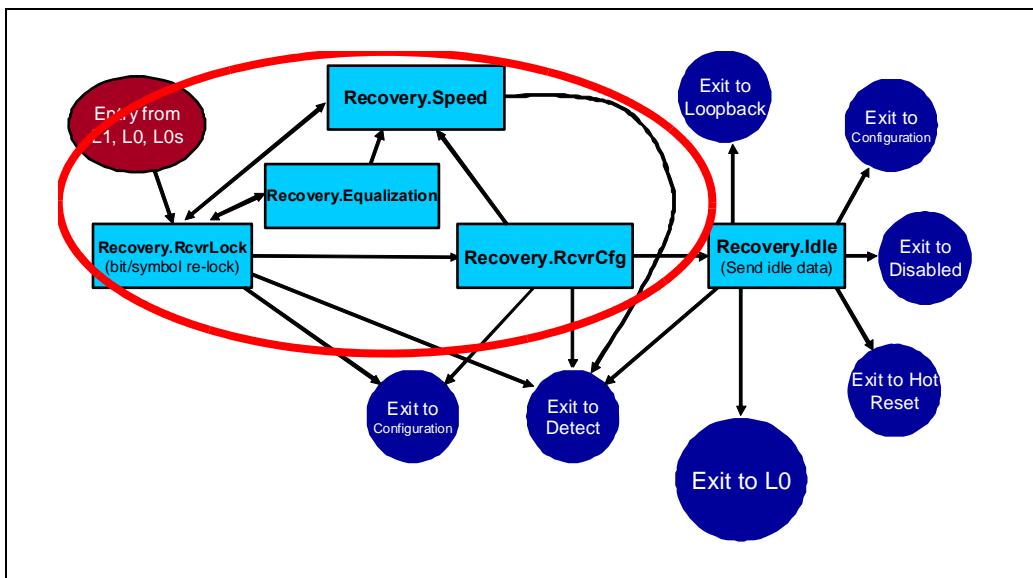
Figure 14-47: TS2 Contents



## Upstream Port Initiates Speed Change

A speed change must be initiated by the Upstream Port (Port facing upstream), and is accomplished by transitioning to the Recovery state. The substates of the Recovery state are shown in Figure 14-48 on page 622 and the part of interest for this discussion is highlighted by the oval. The discussion that follows here is a relatively high-level overview of the entire speed change process and doesn't get into the details of the LTSSM operation. To learn more about that, refer to the discussion called "Recovery State" on page 571.

Figure 14-48: Recovery Sub-States



## Speed Change Example

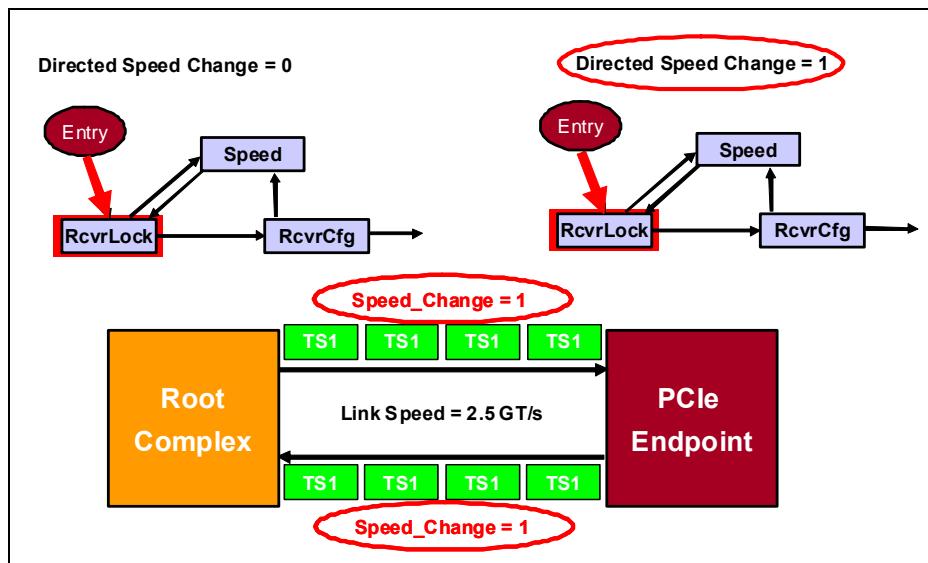
To illustrate the process, consider the speed change example shown in Figure 14-49 on page 623. Note that the Equalization substate has been removed in this example to make the diagrams simpler and easier to follow. The example shows a change from 2.5 GT/s to 5.0 GT/s and so the Equalization substate is not used anyway. A change to 8.0 GT/s would go through the same process but would just add a trip through the Equalization substate at the end of the process. To

## Chapter 14: Link Initialization & Training

learn more about the Equalization process, refer to “Recovery.Equalization” on page 587.

The Endpoint in this example, which can only have an Upstream Port, is shown connected to a Root Complex, which can only have Downstream Ports. Only the Upstream Port can initiate the speed change process, and it does so because its *Directed Speed Change* flag was set earlier based on some hardware-specific conditions. To start the sequence, it changes its LTSSM to the Recovery state, enters the Recovery.RcvrLock substate and sends TS1s with the Speed Change bit set and listing the speeds that it will support, as shown in Figure 14-49 on page 623. When the Downstream Port sees the incoming TS1s, it also changes to the Recovery state and begins sending TS1s back. Since the Speed Change bit was set in the incoming TS1s, that will set the *Directed Speed Change* flag in the Root Port and the outgoing TS1s will also have that bit set. The speed that the Link will attempt to use will be the highest commonly-supported speed so, if a Device wants to use a lower speed it would simply not list the higher speeds as being supported at this time.

Figure 14-49: Speed Change - Initiated

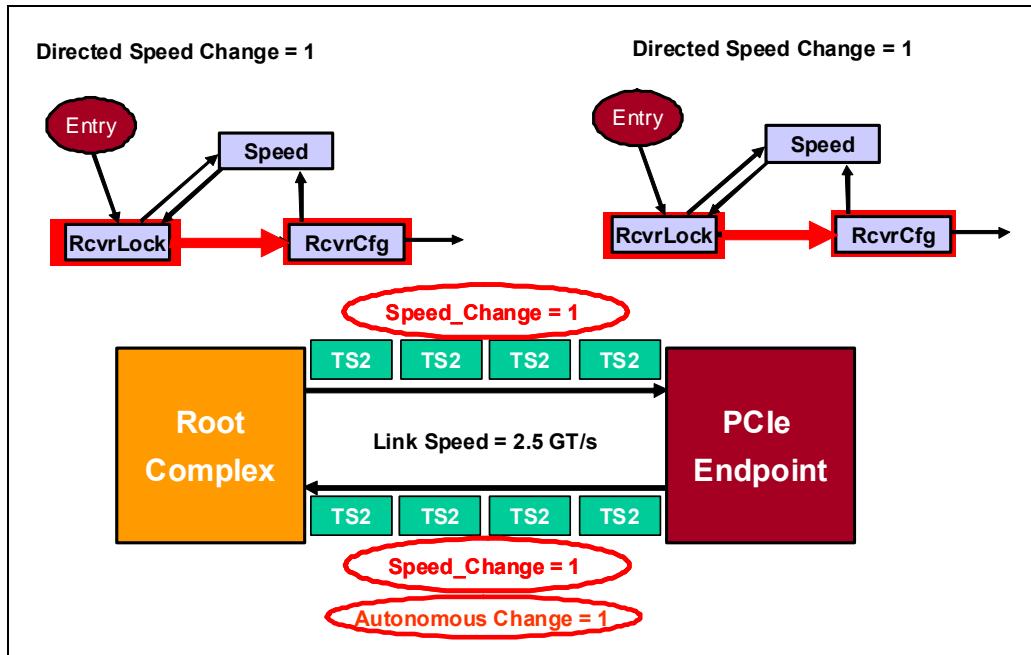


When the Upstream Port detects the TS1s coming back, its state machine changes to the Recovery.RcvrCfg substate and it begins to send TS2s that still have the Speed Change bit set, as illustrated in Figure 14-50 on page 624. These

# PCI Express Technology

TS2s will now also have the Autonomous Change bit set if this change was not caused by a reliability problem on the Link. When the Downstream Port sees incoming TS2s, it also changes to the Recovery.RcvrCfg substate and returns TS2s with the Speed Change bit set. However, the Autonomous Change bit is reserved in the TS2s for Downstream Ports during Recovery.

Figure 14-50: Speed Change - Part 2



Once each Port has seen 8 consecutive TS2s with the Speed Change bit set, they know that the next step will be to go to the Recovery.Speed substate, as shown in Figure 14-51 on page 625. At this point, the Downstream Port needs to register the setting of the Autonomous Change bit in the incoming TS2s. To support this, some extra fields have been added to the PCIe Capability registers.

The status bits for Link bandwidth changes are found in the Link Status register, shown in Figure 14-52 on page 625. Status changes can also be used to generate an interrupt to notify software of these events if the device is capable and has been enabled to do so. This capability is reported by the Link Bandwidth Notification Capable bit, shown in Figure 14-53 on page 626, and enabled by the Interrupt Enable bits in the Link Control register, as shown in Figure 14-54 on

## Chapter 14: Link Initialization & Training

page 626. Note that there are two cases: autonomous and bandwidth management. Autonomous means the change was not caused by a reliability problem, while bandwidth management means it was.

Figure 14-51: Speed Change - Part 3

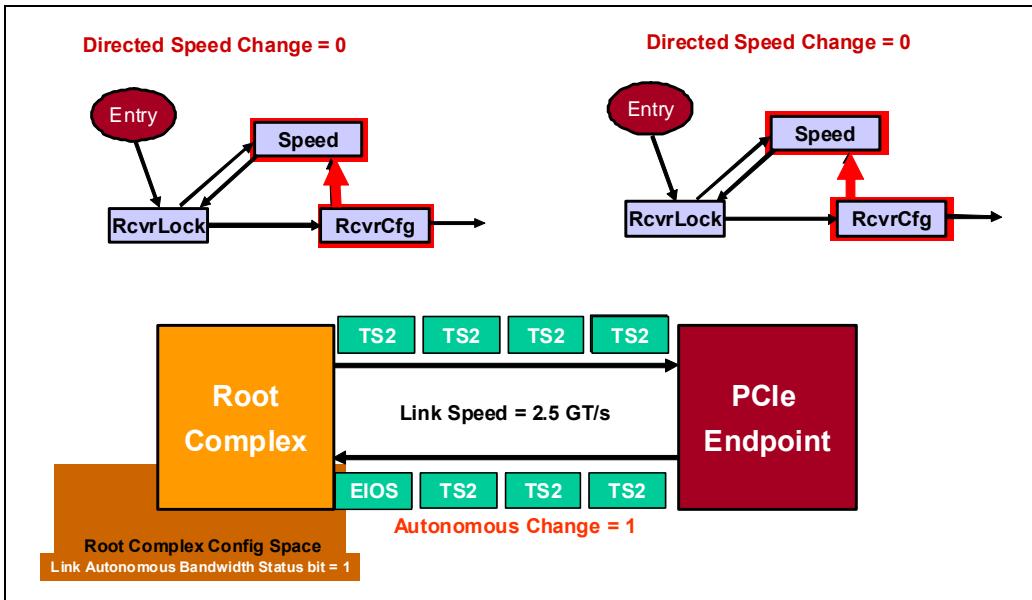
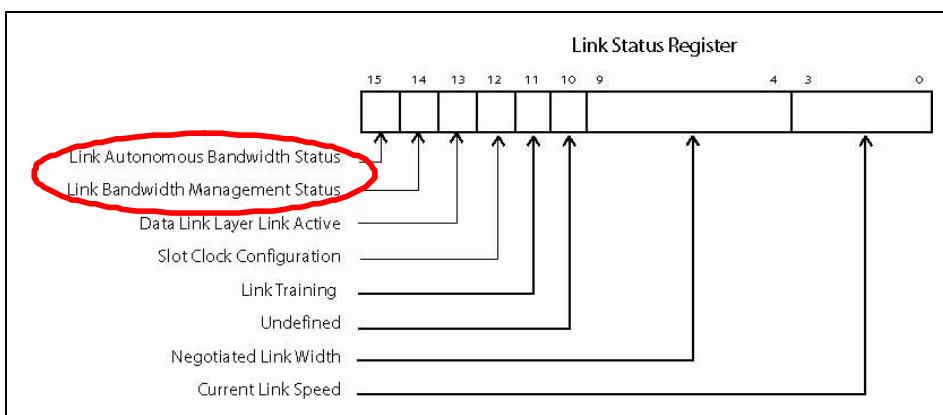


Figure 14-52: Bandwidth Change Status Bits



# PCI Express Technology

---

Figure 14-53: Bandwidth Notification Capability

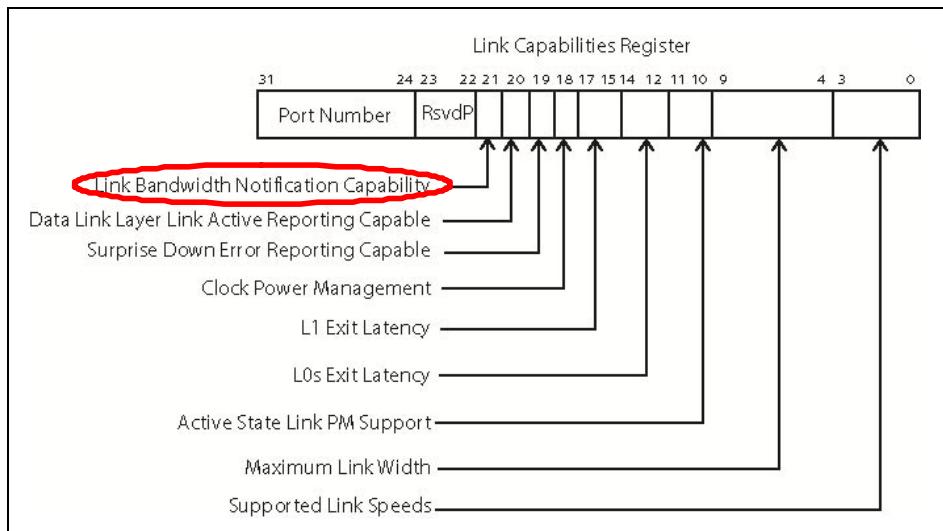
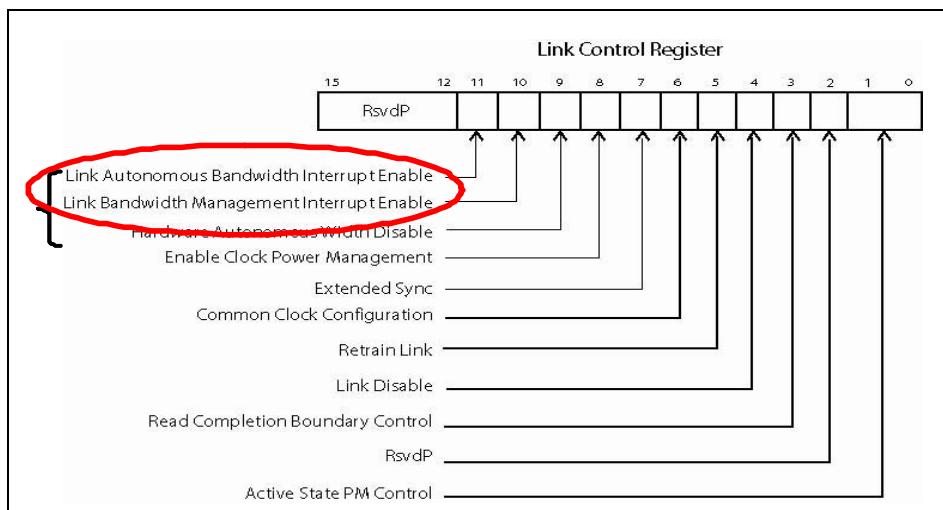


Figure 14-54: Bandwidth Change Notification Bits

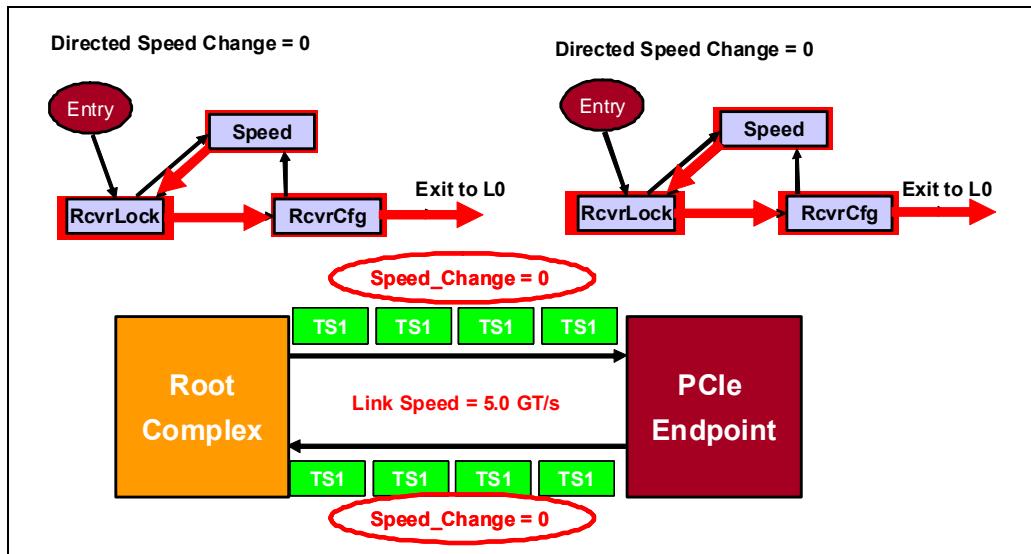


# Chapter 14: Link Initialization & Training

Once the Recovery.Speed substate is reached, the Link is placed into the Electrical Idle condition in both directions and the speed is changed internally. The speed chosen will be the highest commonly-supported speed reported in the Rate ID field of the TS1s and TS2s. In this example, that turns out to be 5.0 GT/s and so the change is made to that speed. After a timeout period, the Link neighbors both transition back to Recovery.RcvrLock and exit Electrical Idle by sending TS1s again, as shown in Figure 14-55 on page 627. When the Upstream Port sees them coming back, it transitions to Recovery.RcvrCfg and begins sending TS2s, much like before. This time, though, the Speed Change bit is not set. Eventually TS2s are seen coming back from the Downstream Port that also don't have the Speed Change bit set, and at that point the state machines transition to the Recovery.Idle on their way back to L0.

If a speed change fails for some reason, a component is not allowed to try that speed or a higher one for at least 200 ms after returning to L0 or until the Link neighbor advertises support for a higher speed, whichever comes first.

Figure 14-55: Speed Change Finish



## Software Control of Speed Changes

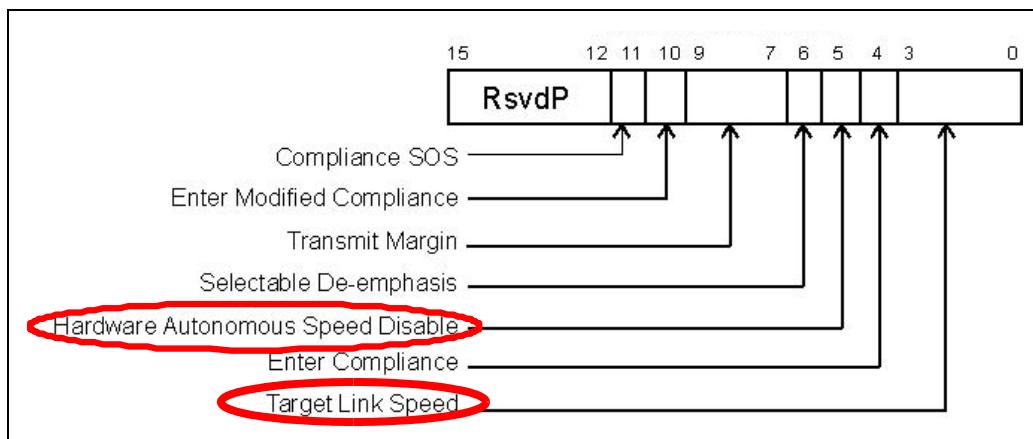
Software is unable to control when hardware makes decisions about changing the speed but can limit or disable this capability. Limiting it is accomplished by setting the Target Link Speed value in the Link Control 2 Register shown in Figure 14-56 on page 628. This acts as the upper bound on the speeds available to

# PCI Express Technology

---

the Upstream Port, which will try to maintain that value or the highest speed supported by both Link neighbors, whichever is lower. Software can also force a particular speed to be used by setting the Target Link Speed in the Upstream component and then setting the Retrain Link bit in the Link Control register, shown in Figure 14-57 on page 629. As mentioned earlier, software is notified of any hardware-based Link speed or width changes by the Link Bandwidth Notification Mechanism. Finally, the speed change mechanism can be disabled by setting the Hardware Autonomous Speed Disable bit.

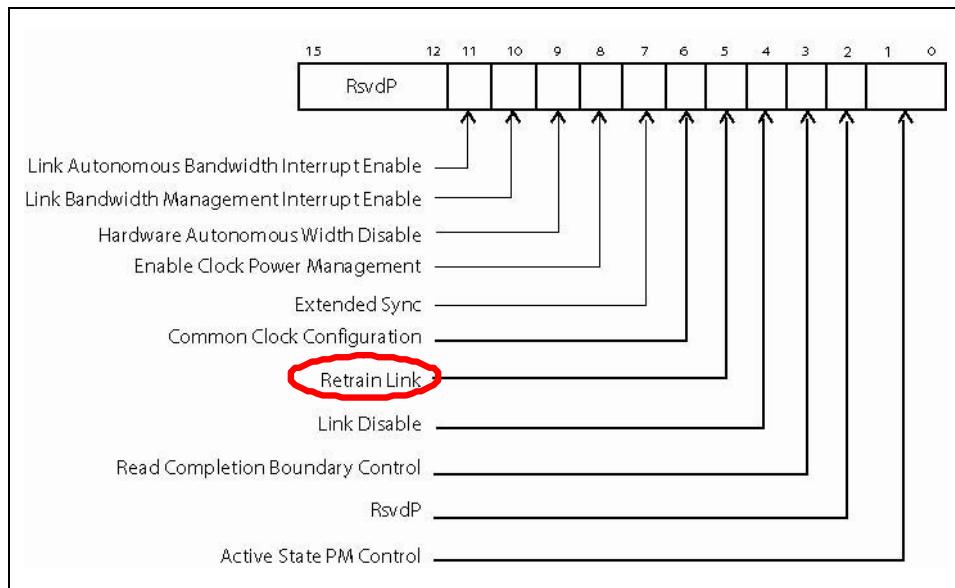
Figure 14-56: Link Control 2 Register



# Chapter 14: Link Initialization & Training

---

Figure 14-57: Link Control Register



---

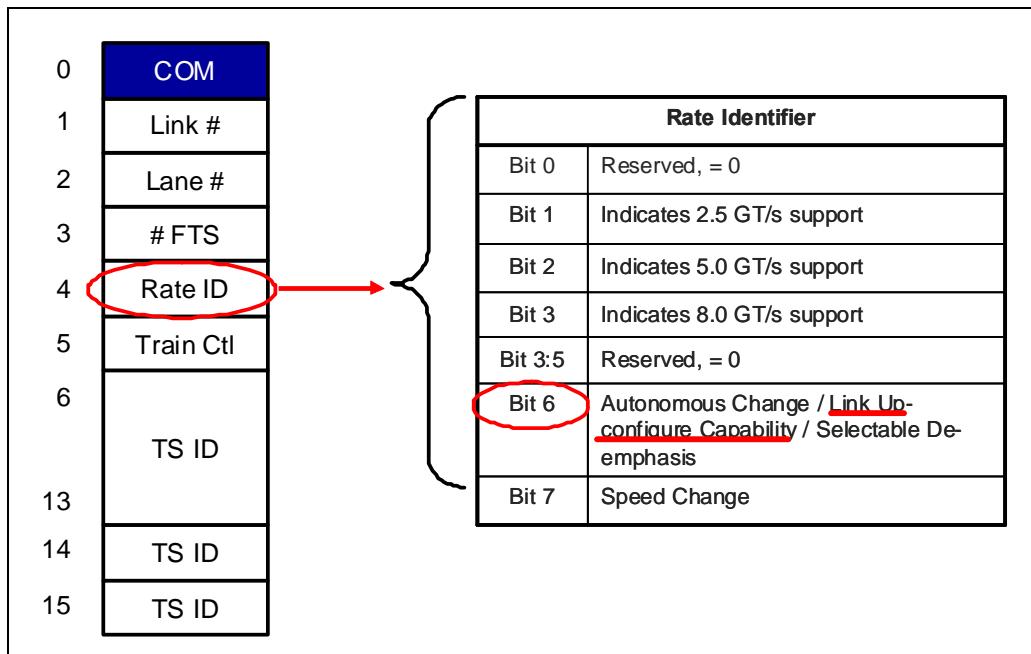
## Dynamic Link Width Changes

The same basic operation for changing the Link speed can also be used to change the Link width, although the sequence is a little more complicated because more LTSSM steps are involved. One thing that's important for software to note before enabling Link width changes is whether the Link neighbor supports recovering from a narrow Link back to a wide Link (called Upconfiguring the Link). Devices report this ability in bit 6 of the Rate ID field of the TS2s they send during training, as shown in Figure 14-58 on page 630. If a component doesn't support this, that would mean that changing to a narrower Link width would be a one-way event and would only be suitable for the case of a reliability problem on the Link.

# PCI Express Technology

---

Figure 14-58: TS2 Contents



---

## Link Width Change Example

Consider the example in Figure 14-59 on page 631 of a Root Port connected to an Endpoint (Gigabit Ethernet Device). Only the Upstream Port will initiate this change, and it begins by going to the Recovery state as before. This time, though, the Speed Change bit is not set. To sort out what the new Link width will be, the Upstream Port will need to tell the Downstream Port to transition from the Recovery state to the Configuration state before going back to L0, as shown in Figure 14-60 on page 631. There are several substates in the Configuration state, and a simplified version of them is shown in Figure 14-61 on page 632. We'll go through the sequence to be clear on how the steps work.

## Chapter 14: Link Initialization & Training

---

Figure 14-59: Link Width Change Example

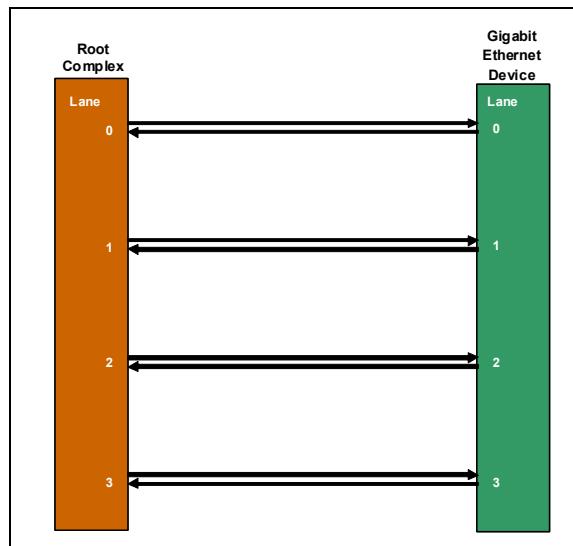


Figure 14-60: Link Width Change LTSSM Sequence

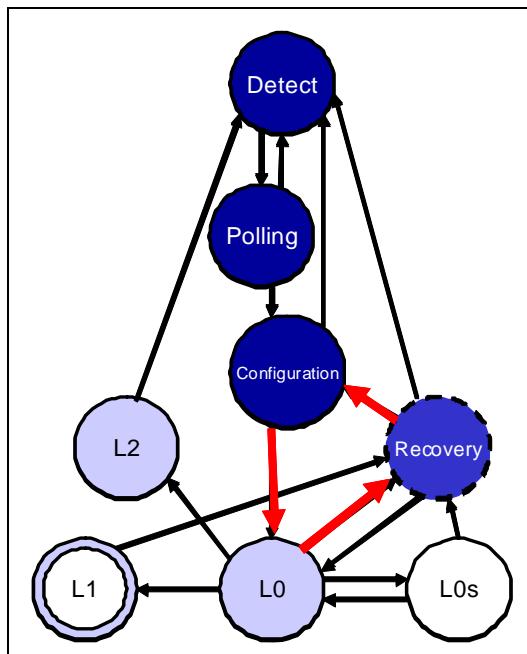
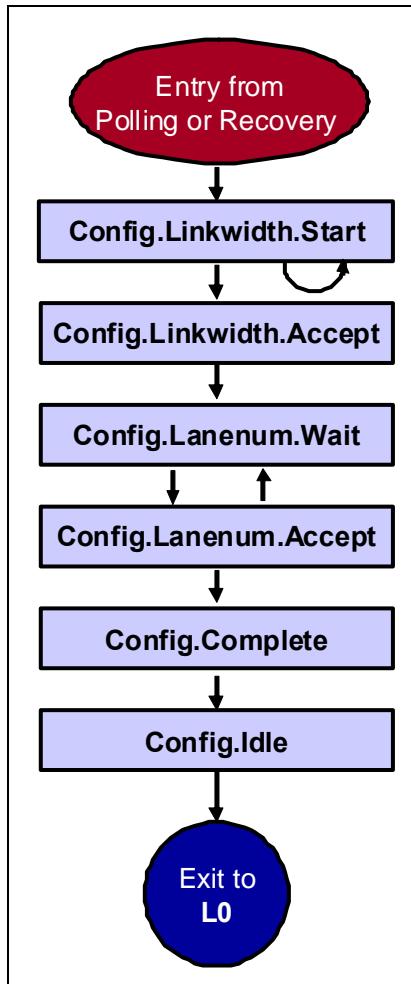


Figure 14-61: Simplified Configuration Substates

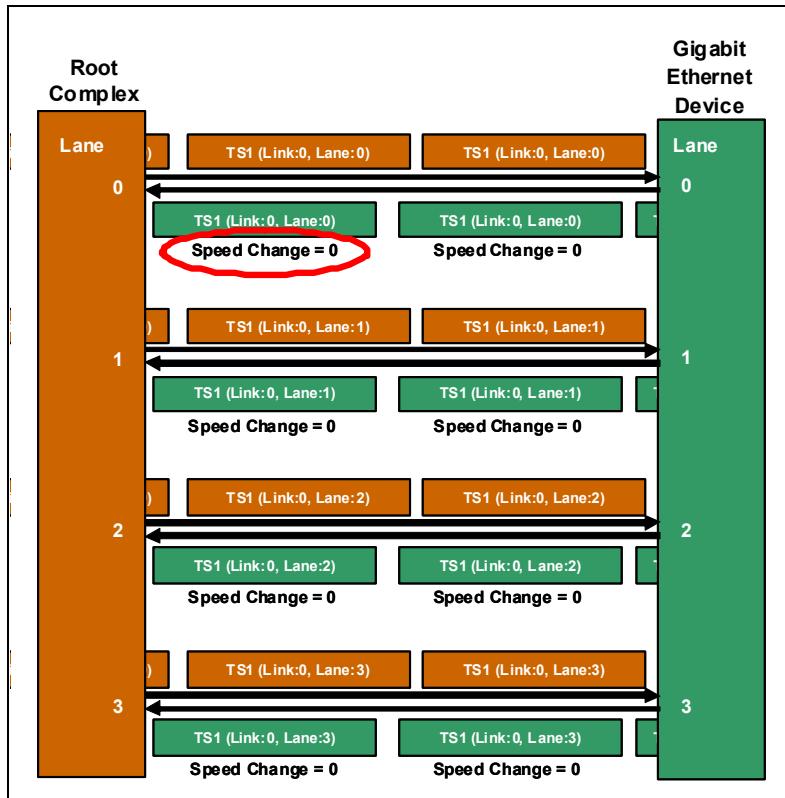


As before, the Upstream Port initiates this process by going to Recovery and sending TS1s. These don't have the Speed Change bit set, as highlighted in the example shown in Figure 14-59 on page 631, where an Ethernet Device initiates this process on its Upstream Port. In response, the Downstream Port sends TS1s back, also with the Speed Change bit cleared. Link and Lane numbers are still shown as being unchanged from the last time the Link was trained. Referring back to Figure 14-48 on page 622, the next state is Recovery.RcvrCfg during which the Link partners exchange TS2s.

## Chapter 14: Link Initialization & Training

---

Figure 14-62: Link Width Change - Start

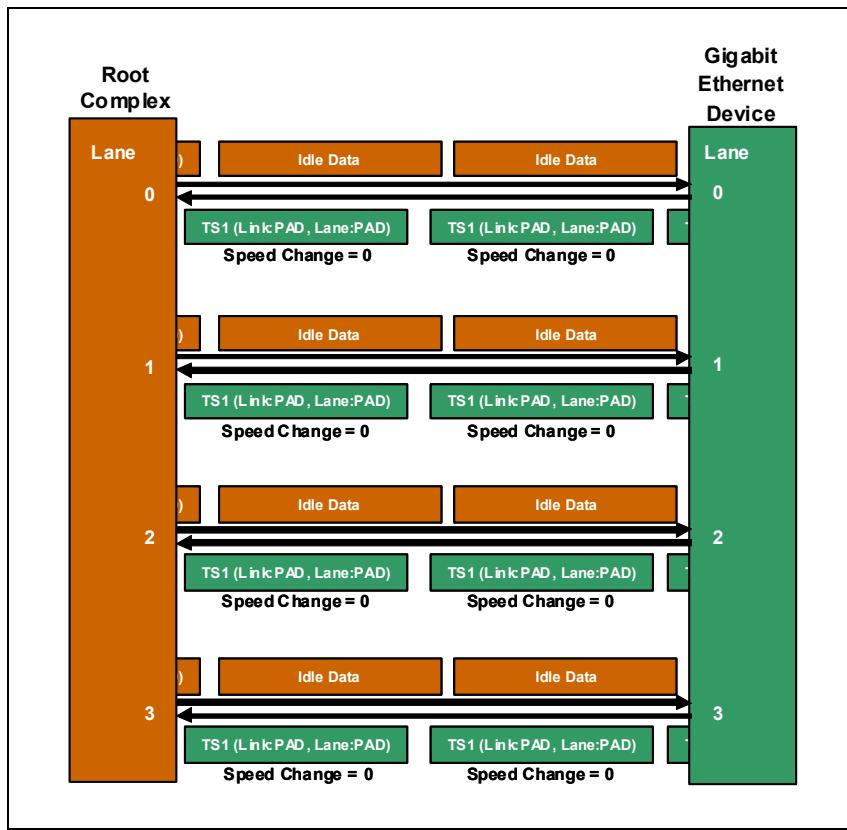


Since a speed change is not requested, the next state is Recovery.Idle. In that state the Ports normally send the logical idle symbols (all zeros) and the Downstream Port does so, as shown in Figure 14-63 on page 634. However, the Upstream Port was directed to change the Link width so it doesn't send the expected Idle symbols. Instead, it sends TS1s with PAD for both the Link and Lane numbers. The Downstream Port recognizes that a previously configured Lane now has a Lane number of PAD, and that causes it to transition to the first Configuration substate: Config.Linkwidth.Start.

# PCI Express Technology

---

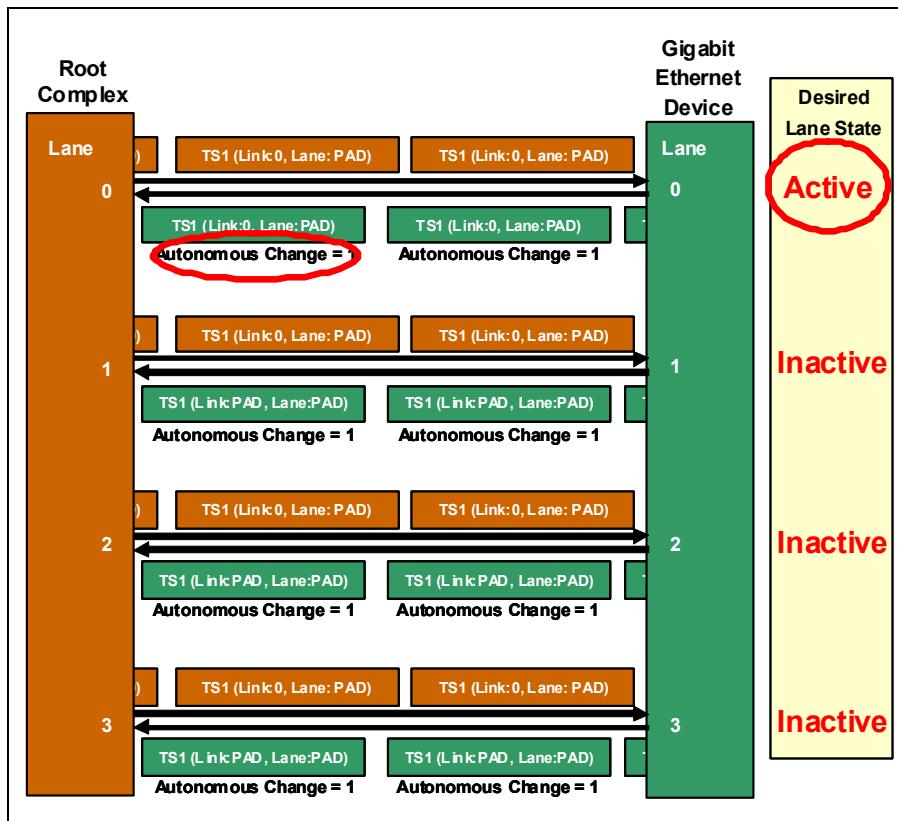
Figure 14-63: Link Width Change - Recovery.Idle



The Downstream Port now initiates the next step by sending TS1s that have the originally negotiated Link number but PAD on all the Lane numbers, as illustrated in Figure 14-64 on page 635. The Upstream Port responds with matching TS1s on the Lanes it wants to have “active”, but with PAD for both Link and Lane numbers on the Lanes it wishes to have inactive. When the Downstream Port sees this response, it transitions to the Config.Linkwidth.Accept substate. Note that the Autonomous Change bit is set for these TS1s.

# Chapter 14: Link Initialization & Training

Figure 14-64: Marking Active Lanes

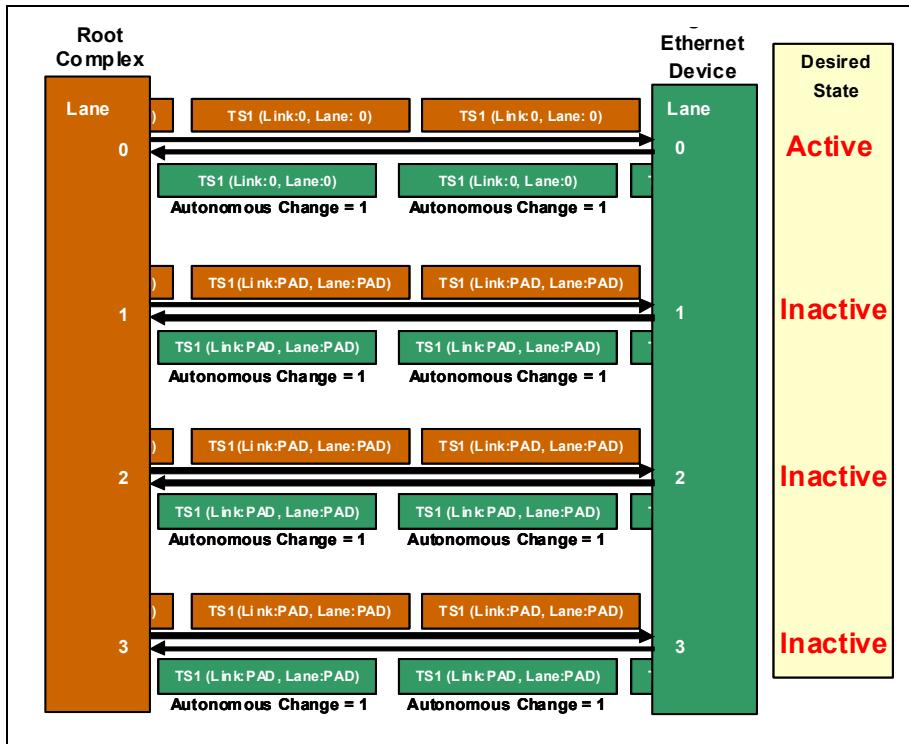


The Root Port responds by changing its TS1s to show Lane numbers that are appropriate for the active Lanes, but PAD for the Link and Lane numbers of all the Lanes that were seen to be inactive. The Upstream Port responds with the same TS1s, as shown in Figure 14-65 on page 636, and the state changes to Config.Lanenum.Accept. At this point, the Root Port updates the status bit to show that an autonomous change was detected and changes to the Config.Complete substate.

# PCI Express Technology

---

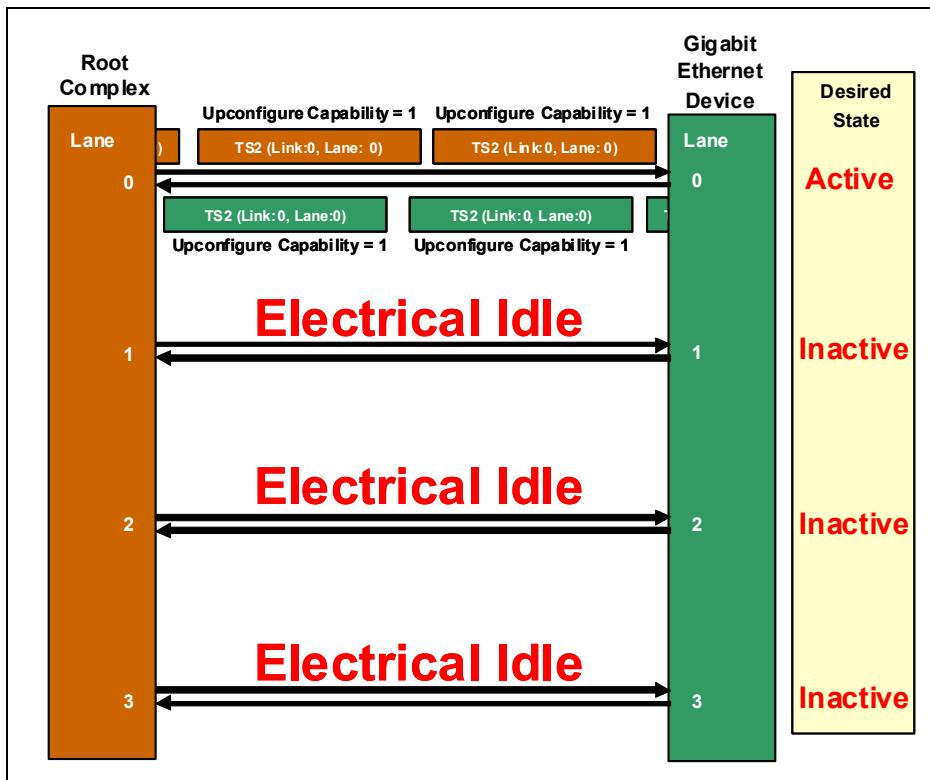
Figure 14-65: Response to Lane Number Changes



In the next step, the Root Port begins to send TS2s on the active Lanes and puts the inactive Lanes into Electrical Idle. Recall that the TS2s report whether a component is “upconfigure capable” and in this example, both Link partners support this capability. The Endpoint sends back the same thing: TS2s on active Lanes and Electrical Idle on inactive Lanes. Seeing that, the Root Port’s state machine changes to Config.Idle and it begins to send Logical Idle on the active Lanes. The Endpoint responds with the same thing and the Link state changes back to L0. The Link is now ready for normal operation, albeit with a reduced bandwidth for power conservation.

## Chapter 14: Link Initialization & Training

Figure 14-66: Link Width Change - Finish

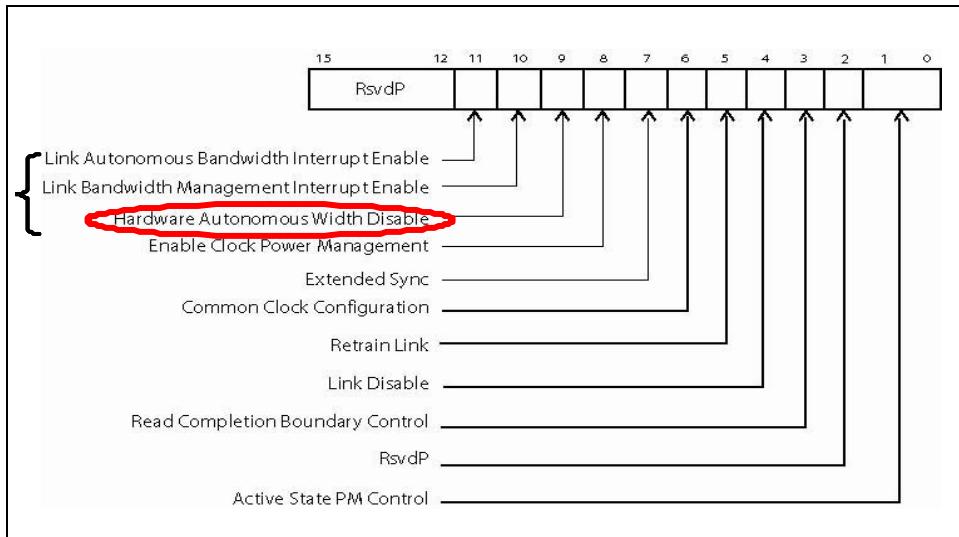


As was the case for dynamic speed changes, software can't initiate Link width changes, but it can disable this mechanism by setting the bit in the Link Control register shown in Figure 14-67 on page 638. Unlike the speed change case, no software mechanism was defined to allow setting a particular Link width.

# PCI Express Technology

---

Figure 14-67: Link Control Register



---

## Related Configuration Registers

Many of the configuration registers that are relevant to Link Initialization and Training have been shown when their contents were described earlier, but it seems good to summarize them here.

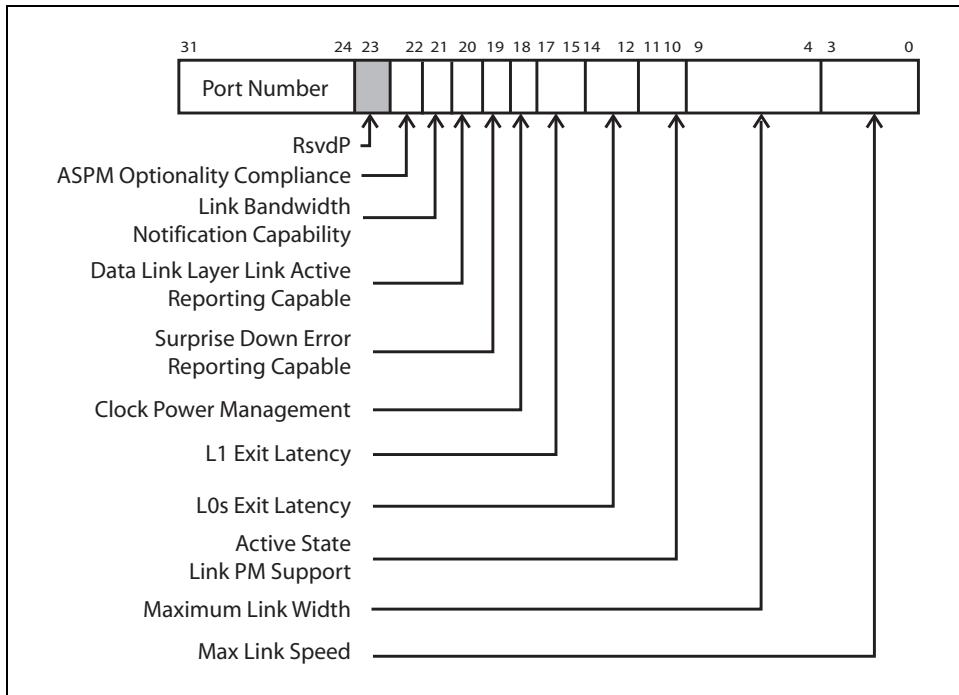
---

## Link Capabilities Register

The Link Capabilities Register is pictured in Figure 14-68 on page 639 and each bit field is described in the subsections that follow.

# Chapter 14: Link Initialization & Training

Figure 14-68: Link Capabilities Register



## Max Link Speed [3:0]

This indicates the maximum Link speed for this port, and is given as a pointer to a bit location in the Link Capabilities 2 register Supported Link Speeds Vector that corresponds to the max Link speed. Defined encodings are:

- 0001b - Supported Link Speeds Vector field bit 0
- 0010b - Supported Link Speeds Vector field bit 1
- 0011b - Supported Link Speeds Vector field bit 2
- 0100b - Supported Link Speeds Vector field bit 3
- 0101b - Supported Link Speeds Vector field bit 4
- 0110b - Supported Link Speeds Vector field bit 5
- 0111b - Supported Link Speeds Vector field bit 6

All other encodings are reserved. Multi-function devices sharing an Upstream Port must report the same value in this field in all Functions. This register is Read Only.

# PCI Express Technology

---

## Maximum Link Width[9:4]

This field indicates the maximum width of the PCI Express Link. The values that are defined are:

- 00 0000b: Reserved
- 00 0001b: x1
- 00 0010b: x2
- 00 0100b: x4
- 00 1000b: x8
- 00 1100b: x12
- 01 0000b: x16
- 10 0000b: x32

All other encodings are reserved. Multi-function devices sharing an Upstream Port must report the same value in this field in all Functions. This register is Read Only.

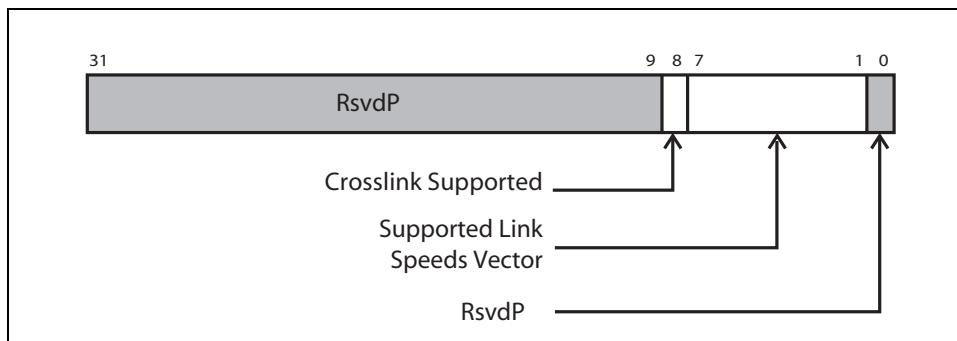
---

## Link Capabilities 2 Register

The Link Capabilities Register is pictured in Figure 14-68 on page 639 and shows the Supported Link Speeds Vector to which the Max Link Speed field in the Link Capabilities register points. The values for this field are:

- Bit 0 = 2.5 GT/s
- Bit 1 = 5.0 GT/s
- Bit 2 = 8.0 GT/s
- Bits 6:3 RsvdP (reserved and preserved).

Figure 14-69: Link Capabilities 2 Register



## Link Status Register

The Link Status Register is pictured in Figure 14-39 on page 597.

### Current Link Speed[3:0]:

This read-only field indicates the current Link speed. The speed will always be 2.5 GT/s when the Link first trains to L0. After that, if a higher commonly-supported speed is available, the LTSSM will go to Recovery and attempt to change to that speed. The values in this field are the same as the Max Link Speed encodings shown in the Link Capabilities register:

- 0001b - Supported Link Speeds Vector field bit 0
- 0010b - Supported Link Speeds Vector field bit 1
- 0011b - Supported Link Speeds Vector field bit 2
- 0100b - Supported Link Speeds Vector field bit 3
- 0101b - Supported Link Speeds Vector field bit 4
- 0110b - Supported Link Speeds Vector field bit 5
- 0111b - Supported Link Speeds Vector field bit 6

All other encodings are reserved.

Note that the value of this field is undefined when the Link is not up (LinkUp = 0b).

### Negotiated Link Width[9:4]

This field indicates the result of link width negotiation. There are seven possible widths, all other encodings are reserved. The defined encodings are:

- 00 0001b: for x1.
- 00 0010b for x2.
- 00 0100b for x4.
- 00 1000b for x8.
- 00 1100b for x12.
- 01 0000b for x16.
- 10 0000b for x32.

All other encodings are reserved. Note that the value of this field is undefined when the Link is not up (LinkUp = 0b).

# PCI Express Technology

---

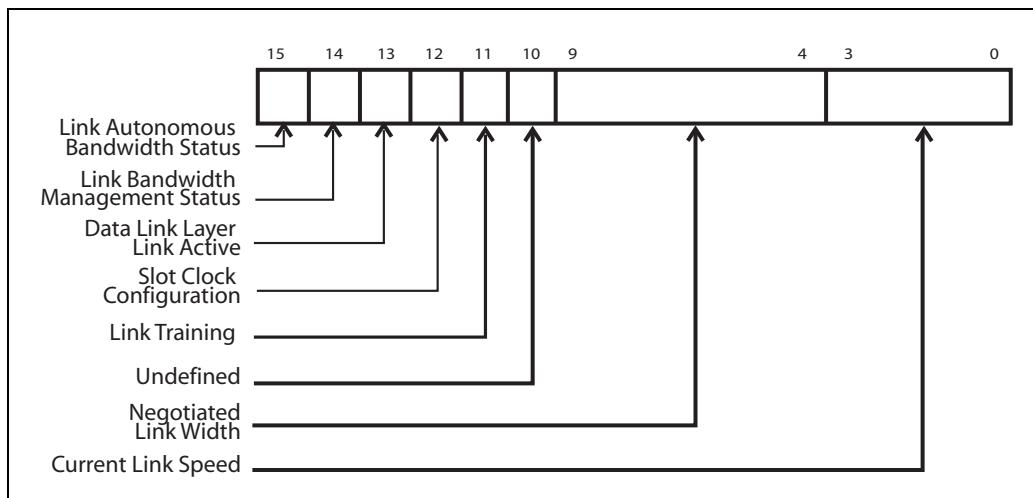
## Undefined[10]

Currently undefined, this bit was previously set by hardware in earlier spec versions when a Link Training Error had occurred. It was cleared when the LTSSM successfully entered L0. The spec states that software can write any value to this bit but must ignore any value read from it.

## Link Training[11]

This read-only bit indicates that the LTSSM is in the process of training. Technically, it means the LTSSM is either in the Configuration or Recovery state, or that the Retrain Link bit has been written to 1b but Link training has not yet begun. This bit is cleared by hardware when the LTSSM exits the Configuration or Recovery state. Since this must be visible to software while Link Training is in progress, it only has meaning for Ports that are facing downstream. Consequently, this bit is not applicable and reserved for Endpoints, bridge Upstream Ports and Switch Upstream Ports. For them, this bit must be hardwired to 0b.

Figure 14-70: Link Status Register



---

## Link Control Register

The Link Control Register is pictured in Figure 14-71 on page 644, and there are three fields in it that are interesting for us here.

# **Chapter 14: Link Initialization & Training**

---

## **Link Disable**

When set to one, the link is disabled. Intuitively, this bit isn't applicable and is reserved for Endpoints, bridge Upstream Ports, and Switch Upstream Ports because it must be accessible by software even when the Link is disabled. When this bit is written, any read immediately reflects the value written, regardless of the Link state. After clearing this bit, software must be careful to honor the timing requirements regarding the first Configuration Read after a Conventional Reset (see “Reset Exit” on page 846).

## **Retrain Link**

This bit allows software to initiate Link re-training whenever it is deemed necessary, as for error recovery. The bit is not applicable to and is reserved for Endpoint devices and Upstream Ports of Bridges and Switches. When set to 1b, this directs the LTSSM to the Recovery state before the completion of the Configuration write Request is returned.

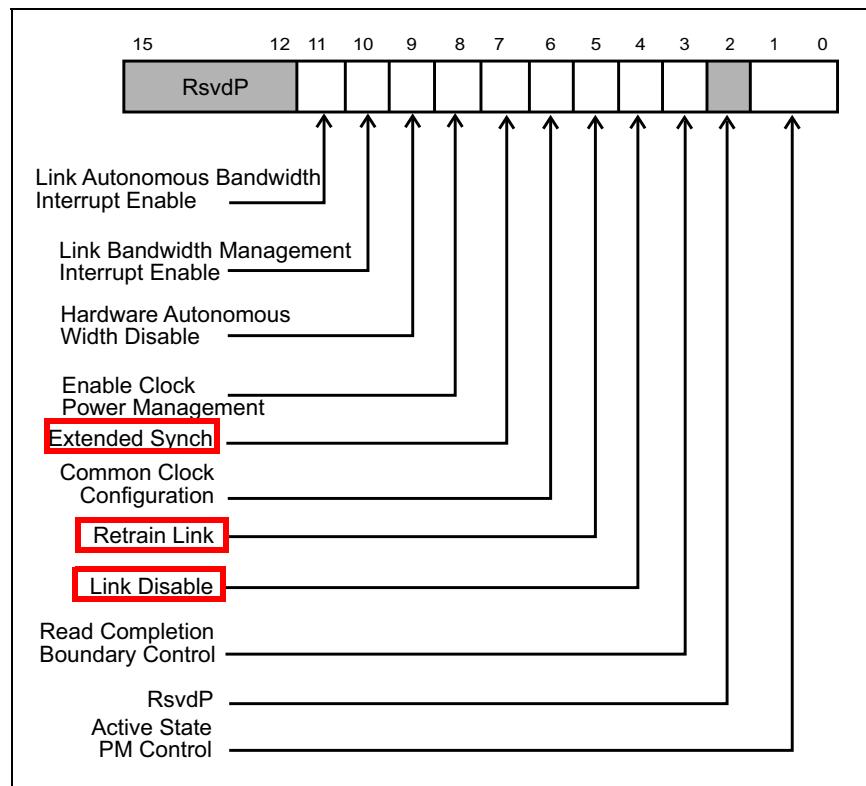
## **Extended Synch**

As it affects training, this bit is used to greatly extend the time spent in two situations, for the purpose of assisting slower external test or analysis hardware to synchronize with the Link before it resumes normal communication. One of these is when exiting L0s, where setting this bit forces the transmission of 4096 FTSs prior to entering L0. The other case is in the Recovery state prior to entering Recovery.RcvrCfg, where it forces the transmission of 1024 TS1s.

# PCI Express Technology

---

Figure 14-71: Link Control Register



# Part Five:

# Additional System Topics



---

# 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#

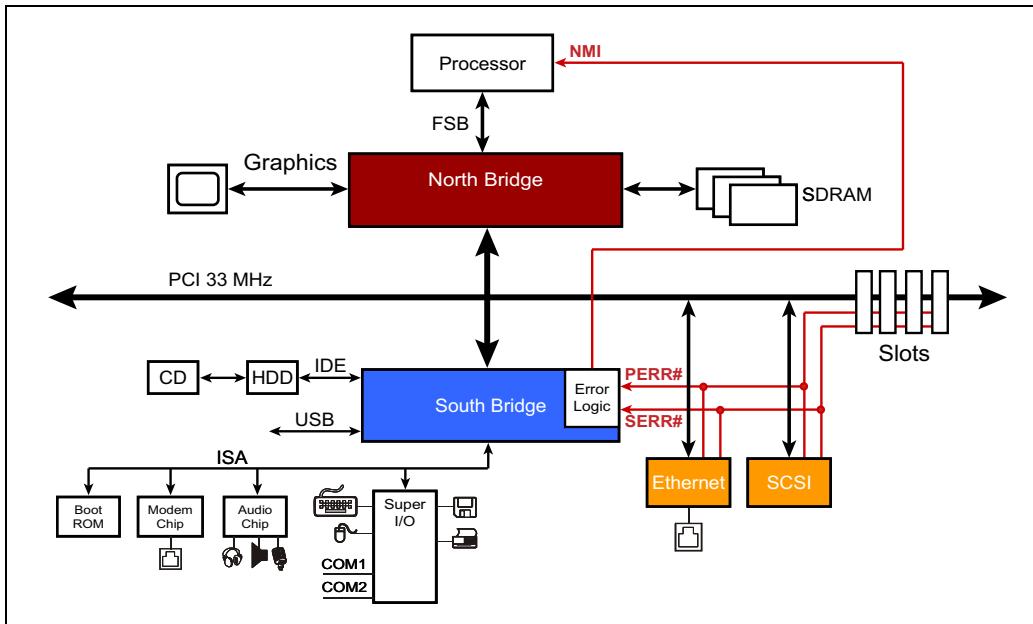
How the errors should be handled was outside the scope of the PCI spec and might include hardware support or device-specific software. As an example, a data parity error on a read from memory might be recovered in hardware by detecting the condition and simply repeating the Request. That would be a safe 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#.

# Chapter 15: Error Detection and Handling

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.

# **Chapter 15: Error Detection and Handling**

---

- PCI Express Capability Registers — these registers will only be useful to newer software that is aware of PCIe, but they provide more error information specifically for PCIe software.

---

## **Advanced Error Reporting (AER)**

This optional error reporting mechanism includes a new and dedicated set of configuration registers that give error handling software more information to work with in diagnosing and recovering from problems. The AER registers are mapped into the extended configuration space and provide much more information about the nature of any errors. See “Advanced Error Reporting (AER)” on page 685 for a detailed description of these registers.

---

## **Error Classes**

Errors fall into two general categories based on whether hardware is able to fix the problem or not, Correctable and Uncorrectable. The Uncorrectable category is further subdivided based on whether software can fix the problem, Non-fatal and Fatal.

- Correctable errors — automatically handled by hardware
- Uncorrectable errors
- Non-fatal — handled by device-specific software; Link is still operational and recovery without data loss may be possible
- Fatal — handled by system software; Link or Device is not working properly and recovery without data loss is unlikely

Based on these classes, error handling software can be partitioned into separate handlers to perform the actions required. Such actions might range from simply monitoring the frequency of Correctable errors to resetting the entire system in the event of a Fatal error. Regardless of the type of error, software may arrange for the system to be notified of all errors to allow tracking and logging them.

---

## **Correctable Errors**

Correctable errors are, by definition, automatically corrected in hardware. They may impact performance by adding latency and consuming bandwidth, but if all goes well, recovery is automatic and fast because it doesn’t depend on software intervention, and no information is lost in the process. These errors aren’t

# PCI Express Technology

---

required to be reported to software, but doing so could allow software to track error trends that might indicate that some devices are showing signs of imminent failure.

---

## Uncorrectable Errors

Errors that can't be automatically corrected in hardware are called Uncorrectable, and these are either Non-fatal or Fatal in severity.

### Non-fatal Uncorrectable Errors

Non-fatal errors indicate that information has been lost but the cause was likely something other than the integrity of a Link or Device. A packet failed somewhere, but the Link continues to function correctly and other packets are unaffected. Since the Link is still working, recovery of the lost information may be possible, but will depend on implementation-specific software to handle it. An example of this error type would be a Completion timeout, in which a Request was sent but no Completion was returned within the allowed time. Somewhere there was an issue, but it could be something as simple as a random bit error within a Switch that caused the Completion to be routed incorrectly. An attempt at recovery for this case could be as simple as re-issuing the Request.

### Fatal Uncorrectable Errors

Fatal errors indicate that a Link or Device has had an operational failure, causing data loss that is unlikely to be recovered. For these cases, resetting at least the failed Link or Device will probably be the first step in any recovery process because it's clearly not operational for some reason. The spec also invites implementation-specific approaches, in which software may attempt to limit the effects of the failure, but it doesn't define any particular actions that should be taken. An example of this type of error would be a receiver buffer overflow, in which case information has been lost because flow control tracking counters have gotten out of sync with each other. Since there's no mechanism to fix this, a reset of this Link will usually be required.

---

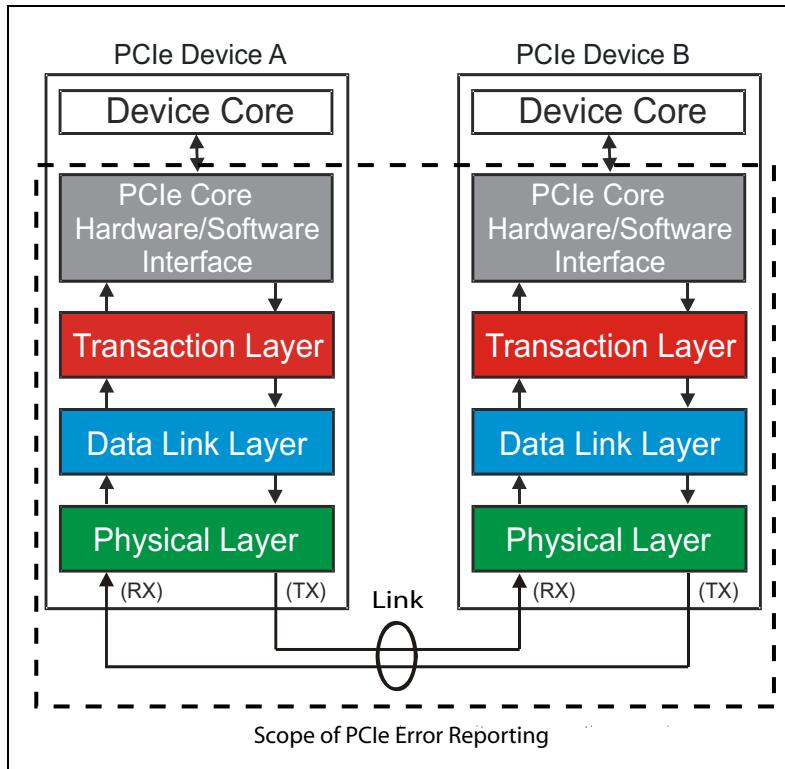
## PCIe Error Checking Mechanisms

The scope of PCIe error checking focuses on errors associated with the Link and packet delivery, as shown in Figure 15-2 on page 653. Errors that don't pertain to Link transmission are not reported through PCIe error-handling mechanisms and would need proprietary methods to report them, such as device-specific

# Chapter 15: Error Detection and Handling

interrupts. Each layer of the interface includes error checking capabilities, and these are summarized in the sections that follow.

Figure 15-2: Scope of PCI Express Error Checking and Reporting



## CRC

Before diving into error handling as it relates to the layers, it will help to first discuss the concept of CRC (Cyclic Redundancy Check) because it's an integral part of PCIe error checking. A CRC code is calculated by the transmitter based on the contents of the packet and adds it to the packet for transmission. The CRC name is derived from the fact that this *check* code (calculated from the packet to check for errors) is *redundant* (adds no information to the packet), and is derived from *cyclic* codes. Although a CRC doesn't supply enough information to do automatic error correction the way ECC (Error Correcting Code) can, it does provide robust error detection. CRCs are also commonly used in serial transports because they're good at detecting a string of incorrect bits.

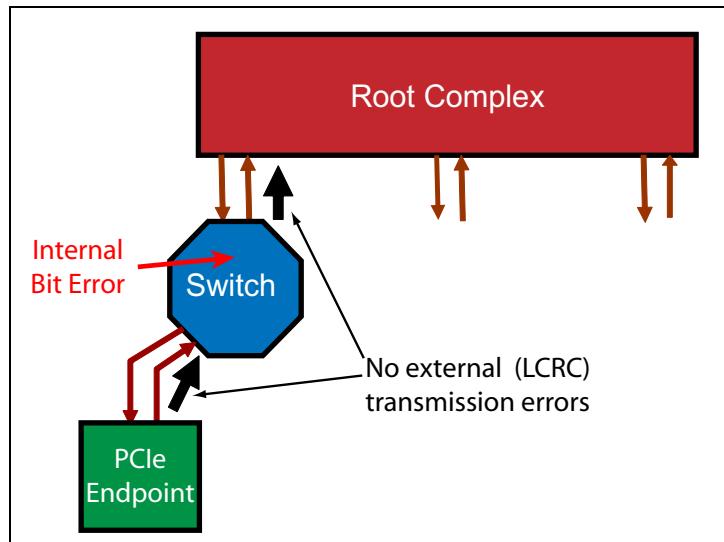
# PCI Express Technology

---

CRCs have two different usage cases in PCIe. One is the mandatory LCRC (Link CRC) generated and checked in the Data Link Layer for every TLP that goes across a Link. It's intended to detect transmission errors on the Link.

The second is the optional ECRC (End-to-end CRC) that's generated in the Transaction Layer of the sender and checked in the Transaction Layer of the ultimate target of the packet. This is intended to detect errors that might otherwise be silent, such as when a TLP passes through an intermediate agent like a Switch, as shown in Figure 15-3 on page 654. In this illustration, the packet arrived safely on the downstream port of the Switch but while it was being stored or processed within the Switch a bit error occurred. The LCRC only protects TLPs while on the Link. Once the Data Link Layer of the Ingress Port checks the LCRC, it removes it from the packet because a new LCRC will be calculated (which will include the new Sequence Number) at the Egress Port. This means that the packet is unprotected while inside the Switch. This is the purpose of having an ECRC. It is calculated at the originating device and is not removed or recalculated by intermediate devices. So if the target device is checking the ECRC and sees a mismatch, then there must have been an error somewhere along the way even though no LCRC error was seen. Note that using the ECRC requires the presence of the optional Advanced Error Reporting registers, since they contain the bits to enable this functionality.

Figure 15-3: ECRC Usage Example



## Error Checks by Layer

Different aspects of an incoming packet are checked in the different layers at the Receiver. Some error checking is listed as optional. For those cases, if the error occurs but the designer has chosen not to implement that form of checking, it will not be detected.

### Physical Layer Errors

A packet arriving at the Receiver arrives at the Physical Layer first. There are a few things that must be checked at this level and others that may optionally be checked. Link training also takes place at this layer, and a variety of problems may arise during that process but those and other details of the Physical Layer are covered in Chapter 14, entitled "Link Initialization & Training," on page 505. In summary, though, Physical Layer errors, also called Receiver Errors or Link Errors, include the following cases:

- When using 8b/10b, checking for decode violations (checking required)
- Framing violations (optional for 8b/10b, required for 128b/130b)
- Elastic buffer errors (checking optional)
- Loss of symbol lock or Lane deskew (checking optional)

If a TLP was in progress when a Receiver Error was detected, it is discarded. To resolve the error, the Data Link Layer is signaled to send a NAK if one isn't already pending.

### Data Link Layer Errors

After the Physical Layer, incoming packets go next into the Data Link Layer, where they are checked for several possible problems. The details of these conditions can be found in Chapter 10, entitled "Ack/Nak Protocol," on page 317. In summary, the errors are:

- LCRC failure for TLPs
- Sequence Number violation for TLPs
- 16-bit CRC failure for DLLPs
- Link Layer Protocol errors

As with the Physical Layer, if a TLP was in progress when an error is seen, the TLP is discarded and a NAK is scheduled if one isn't already pending.

There are some Data Link Layer errors to watch for at the transmitter, too, including REPLAY\_TIMER expiring and the REPLAY\_NUM counter rolling over. A timeout is handled by replaying the contents of the Replay Buffer and

# PCI Express Technology

---

incrementing the REPLAY\_NUM counter. The timer and counter are reset whenever an ACK or NAK arrives at the transmitter that indicates forward progress has been made (meaning it results in clearing one or more TLPs from the Replay Buffer). But if an Ack or Nak isn't received quickly enough, the time-out condition is seen which will result in a replay.

## Transaction Layer Errors

Lastly, if incoming TLPs pass all the checks at the Physical and Data Link Layers, they will finally reach the Transaction Layer, where they are checked for:

- ECRC failure (checking optional)
- Malformed TLP (error in packet format)
- Flow Control Protocol violation
- Unsupported Requests
- Data Corruption (poisoned packet)
- Completer Abort (checking optional)
- Receiver Overflow (checking optional)

As with the Data Link Layer, there are some error checks at the transmitter Transaction Layer, too, such as:

- Completion Timeouts
- Unexpected Completion (Completion does not match pending Request)

---

## Error Pollution

A problem can arise if a device sees several problems for the same transaction. This could result in several errors getting reported (referred to as "Error Pollution"). To avoid this, reported errors are limited to only the most significant one. For example, if a TLP has a Receiver Error at the Physical Layer, it would certainly be found to have errors at the Data Link Layer and Transaction Layers, too, but reporting them all would just add confusion. What is most relevant is reporting the first error that was seen. Consequently, if an error is seen in the Physical Layer, there's no reason to forward the packet to the higher layers. Similarly, if an error is seen in the Data Link Layer, then the packet won't be forwarded to the Transaction Layer. Offending packets at one level are not forwarded to the next level but are dropped.

Still, multiple errors may be seen for the same packet at the Transaction Layer. Only the most significant one should be reported in the order of priority as defined by the spec. Transaction Layer error priority from highest to lowest is:

# **Chapter 15: Error Detection and Handling**

---

- Uncorrectable Internal Error
- Receiver Buffer Overflow
- Flow Control Protocol Error
- ECRC Check Failed
- Malformed TLP
- AtomicOp Egress Blocked
- TLP Prefix Blocked
- ACS (Access Control Services) Violation
- MC (Multi-cast) Blocked TLP
- UR (Unsupported Request), CA (Completer Abort), or Unexpected Completion
- Poisoned TLP Received

As an example, a TLP might experience an ECRC fault caused by a corrupted header. Since something was corrupted within the packet, it might also be seen as Malformed or possibly as an Unsupported Request. The ECRC fault is the highest priority, since it means that the header contents may have been corrupted, and due to this, there is no point in reporting errors that depend on those contents.

---

## **Sources of PCI Express Errors**

Rather than consider all of the error conditions individually, it will be helpful to group them into common areas.

---

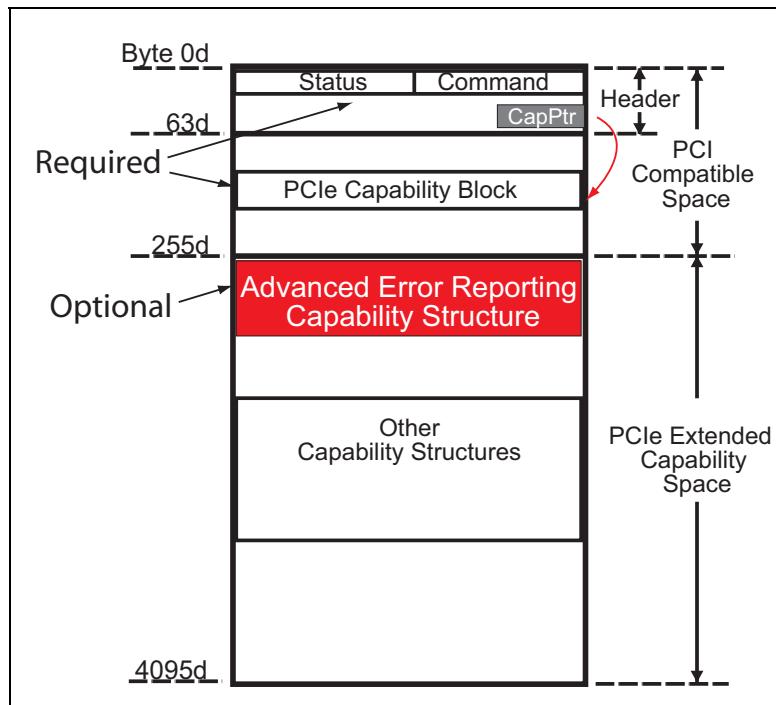
### **ECRC Generation and Checking**

As mentioned earlier, ECRC generation and checking requires the optional Advanced Error Reporting configuration register structure to be present, as shown in Figure 15-4 on page 658. Configuration software checks for this capability register to determine whether ECRCs are supported in a Function. If it is, a write to the Error Capability and Control register can be used to enable it.

# PCI Express Technology

---

Figure 15-4: Location of Error-Related Configuration Registers



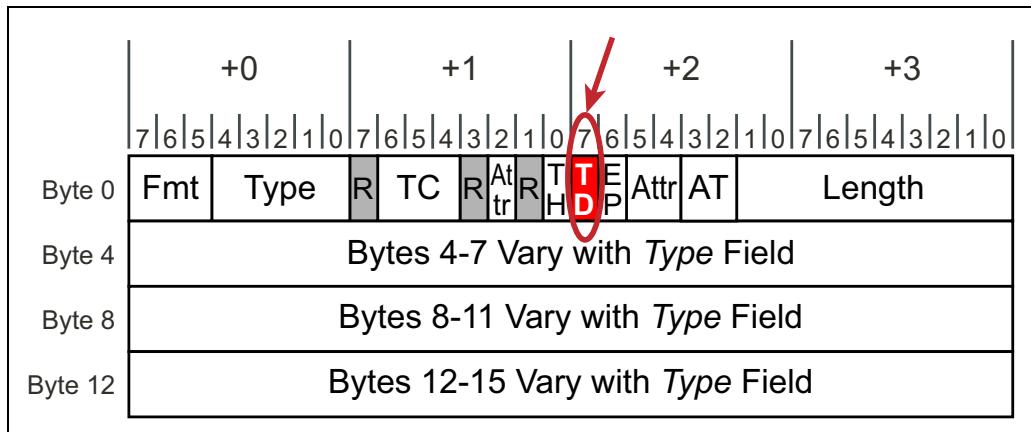
A device enabled to generate ECRCs originates a TLP (Request or Completion), computes the 32-bit ECRC based on the header and data portions of the packet and adds it to the end of the packet. The ECRC is called “end-to-end” because the intent is that it will be generated at the TLP’s origin and never stripped off or regenerated by any intermediate device along its path. Switches in the path between the originating and receiving devices are allowed to check and report ECRC errors but aren’t required to do so. Whether or not there is an error, a Switch must still forward the packet unaltered so that the ultimate target device can evaluate the ECRC and take appropriate steps. If a Switch is acting as the originator or recipient of the TLP it can participate like an ordinary device in ECRC generation and checking. For more on the topic of how a Switch is allowed to report such errors, see “Advisory Non-Fatal Errors” on page 670.

# Chapter 15: Error Detection and Handling

## TLP Digest

If the optional ECRC capability is enabled, a special bit called TD (TLP Digest) is set in the header to indicate that it's present at the end of the packet (the ECRC is also called the Digest). The TD bit in the packet header is shown in Figure 15-5 on page 659. The spec emphasizes that this bit must be treated with special care when forwarding a TLP because if it's missing but the ECRC is present, or vice-versa, then the packet will be considered Malformed.

Figure 15-5: TLP Digest Bit in a Completion Header



## Variant Bits Not Included in ECRC Mechanism

The ECRC is calculated based on the contents of the header and data. Since these are not expected to change, the result should be the same when the check is performed at the receiver. However, it turns out that two header bits can legally change while the packet is in flight: bit 0 of the Type field, and the EP bit. Bit 0 of the Type field can change in Configuration Requests for the simple reason that the Request will be Type 1 until it has reached its destination bus, and then it will become Type 0. That involves changing bit 0 of the Type field. The EP bit can also be legally changed by intermediate devices if they detect a data error. For example, if a Switch forwards a TLP but it suffers an internal error of some kind that corrupts the data, setting the EP bit as it goes out the Egress Port is one way to report the error (known as error forwarding or data poisoning).

Since these two bits can change while the packet is in flight they are called "variant bits" and cannot be used in the generation or checking of ECRC. Instead, their values are always assumed to be 1b for ECRC generation and checking instead of using the actual values. That way the ECRC doesn't depend on them and will be correctly evaluated.

# PCI Express Technology

---

The actions taken when an ECRC error is detected are beyond the scope of the spec, but the possible choices will depend on whether the error is found in a Request or a Completion.

- **ECRC in Request** — Completers that detect an ECRC error must set the ECRC error status bit. They may also choose not to return a Completion for this Request, resulting in a Completion timeout at the Requester, whose software might then choose to reschedule the Request.
- **ECRC in Completion** — Requesters that detect an ECRC error must set the ECRC error status bit. Besides the standard error reporting mechanism, they may also choose to report the error to their device driver with a Function-specific interrupt. As before, the software might decide to reschedule the failed Request.

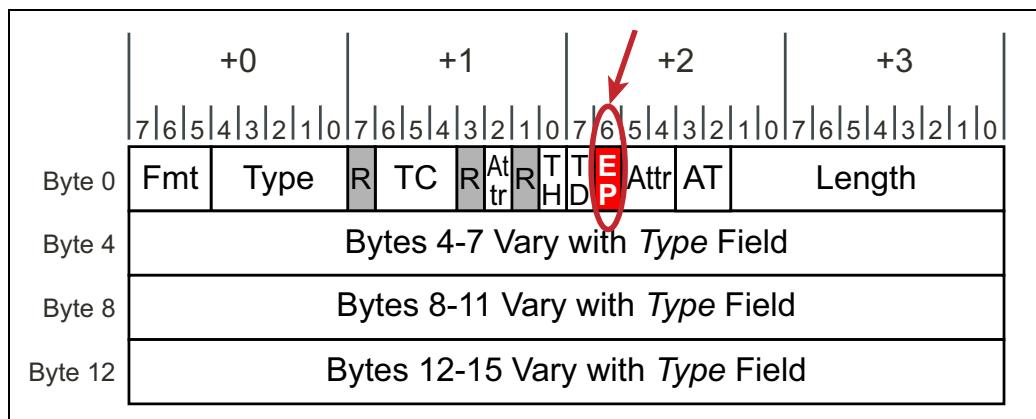
In either case, an Uncorrectable Non-fatal error Message may be sent to the system. If so, the device driver would probably be accessed to check the status bits in the *Uncorrectable Error Status Register* and learn the nature of the error. If possible, the failed Request may be rescheduled, but other steps might be needed.

---

## Data Poisoning

Data poisoning, also called Error Forwarding, provides an optional way for a device to indicate that the data associated with a TLP is corrupted. In these cases, the EP (Error Poisoned) bit in the packet header is set to indicate the error. The EP bit is shown in Figure 15-6 on page 660.

Figure 15-6: The Error/Poisoned Bit in a Completion Header



## Chapter 15: Error Detection and Handling

---

Anytime data is transferred, such as in write Requests or Completions with data, corruption of that data could happen which needs to be reported to the target device. In each of these cases, the packet can be forwarded to the recipient but marked as having bad data by the EP bit in the header. The thoughtful reader may wonder why one might want to send data that is already known to be bad. As it happens, there are some cases where it's useful:

1. If a Request results in a Completion returned with data, but that data encountered an error as it was gathered from the target (like a parity or ECC failure in memory), then what is the best way to report it? One approach would be not to send the Completion at all but, if the error isn't reported in some other way, the system only sees a Completion timeout at the Requester. That response isn't very helpful because any number of problems might result in that outcome.  
If, on the other hand, the Completion is delivered with the poisoned bit set, then at least the Requester can see that the round-trip path to the Completer must have been working correctly. Therefore, the problem must have occurred internally to the Completer or else in a Switch that was in the path. What steps will be taken will be implementation specific, but more is known about what must have gone wrong than if the Completion simply timed out.
2. It can be used to report an intermediate problem. If a data payload is corrupted while passing through a Switch, the packet can still be forwarded with the EP bit set to indicate the problem.
3. It may be that the target device can accept the data with errors. As an example, an audio output device needs to receive a timely data stream to work well. If incoming data has an error, the consequences are small (glitch in the audio output) and the time to recover would be long enough to cause a noticeable delay, so it can be better to take it as is rather than attempting recovery of the data.
4. A target device might have a means of correcting the data. The data might be directly recoverable, or the target might have a means of re-creating parts of it, or have some other means of working around the problem.

The spec states that data poisoning applies only to the data payload associated with a packet (such as Memory, Configuration, or I/O writes and Completions) and never to the contents of the TLP header. Consequently, a receiver's behavior is undefined if it sees a poisoned packet (EP=1) with no payload (like a poisoned memory read). Poisoning can only be done at the Transaction Layer of a device; the Data Link Layer does not examine or affect the contents of the TLP header.

Error forwarding support is stated to be optional for transmitters, and the absence of such a statement for receivers implies that it's not optional for them.

# PCI Express Technology

---

If a transmitter supports it, it's enabled with the Parity Error Response bit in the legacy Command register. That's because a Poisoned packet is roughly analogous to a parity error in PCI, since that's how PCI reports bad data. Receipt of a poisoned packet may be reported to the system with an error Message if enabled and, if the optional Advanced Error Reporting registers are present, will also set the Poisoned TLP status bit.

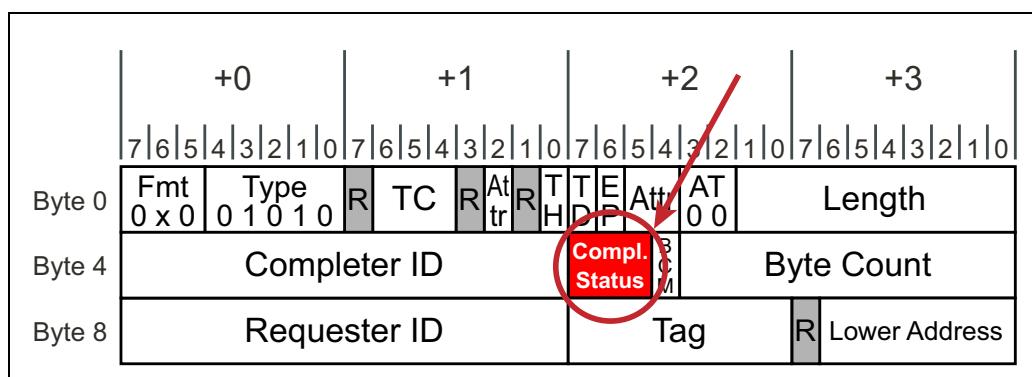
As one might expect, poisoned writes to control locations are not allowed to modify the contents in the target. Examples given in the spec are Configuration writes, IO or memory writes to control registers, and AtomicOps. Switches that receive poisoned packets must forward them unchanged to the destination port although, if they've been enabled to do so, they must report this packet as an error to help software determine where the error happened. Completers that receive a poisoned non-posted Request are expected to return a Completion with a status of UR (Unsupported Request).

---

## Split Transaction Errors

A variety of failures can occur during a split transaction associated with non-posted requests. PCIe defines a status field within the Completion header that allows the Completer to report some errors back to the Requester. Figure 15-7 on page 662 illustrates the location of this field in a completion header and Table 15-1 on page 663 gives the possible values. As the table shows, only four encodings are defined, two of which represent error conditions.

Figure 15-7: Completion Status Field within the Completion Header



# Chapter 15: Error Detection and Handling

---

Table 15-1: Completion Code and Description

Status Code	Completion Status Definition
000b	Successful Completion (SC)
001b	Unsupported Request (UR) - error
010b	Configuration Request Retry Status (CRS)
011b	Completer Abort (CA) - error
100b - 111b	Reserved

## Unsupported Request (UR) Status

If a receiver doesn't support a Request, it returns a Completion with UR status. The spec defines a number of conditions that could result in a UR status. Some examples are:

- Request type not supported (example: IO Request to native Endpoint or MRdLk to native Endpoint)
- Message with unsupported or undefined message code
- Request does not reference address space mapped to the device
- Request address isn't mapped within a Switch Port's address range
- Poisoned write Request (EP=1) targets an I/O or Memory-mapped control space in the Completer. Such Requests must not be allowed to modify the location and are instead discarded by the Completer and reported with a Completion having a UR status.
- A downstream Root or Switch Port receives a configuration Request targeting a device on its Secondary Bus that doesn't exist (e.g. a device with a non-zero device number, unless ARI is enabled). The Port must terminate the Request and return a Completion with UR status because the downstream Device number is required to be zero (unless ARI, Alternative Routing-ID Interpretation, is enabled).
- Type 1 configuration Request is received at an Endpoint.
- Completion using a reserved Completion Status field encoding must be interpreted as UR.
- A function in the D1, D2, or D3<sub>hot</sub> power management state receives a Request other than a configuration Request or Message.
- A TLP without the No Snoop bit set in its header is routed to a port that has the Reject Snoop Transactions bit set in its VC Resource Capability register.

## Completer Abort (CA) Status

Several circumstances can occur that could result in a Completer returning this CA status to the Requester. Some examples are:

- Completer receives a Request that it cannot complete without violating its programming rules. For example, some Functions may be designed to only allow accesses to some registers in a complete and aligned manner (e.g. a 4-byte register may require a 4-byte aligned access). Any attempt to access one of these registers in a partial or misaligned fashion (e.g. reading only two bytes of a 4-byte register) would fail. Such restrictions are not violations of the spec, but rather legal constraints associated with the programming interface for this Function. Access to such a Function is based on the expectation that the device driver understands how to access its Function.
- Completer receives a Request that it cannot process because of some permanent error condition in the device. For example, a wireless LAN card that won't accept new packets because it can't transmit or receive over its radio until an approved antenna is attached.
- Completer receives a Request for which it detects an ACS (Access Control Services) error. An example of this would be a Root Port that implements the ACS registers and has ACS Translation Blocking enabled. If a memory Request is seen on that Port with anything other than the default value in the AT field, it will be an ACS violation.
- PCIe-to-PCI Bridge may receive a Request that targets the PCI bus. PCI allows the target device to signal a target abort if it can't complete the Request due to some permanent condition or violation of the Function's programming rules. In response, the bridge would return a Completion with CA status.

A Completer that aborts a Request may report the error to the Root with a Non-fatal Error Message and, if the Request requires a Completion, the status would be CA.

## Unexpected Completion

When a Requester receives a Completion, it uses the transaction descriptor (Requester ID and Tag) to match it with an earlier Request. In rare circumstances, the transaction descriptor may not match any previous Request. This might happen because the Completion was mis-routed on its journey back to the intended Requester. An Advisory Non-fatal Error Message can be sent by the device that receives the unexpected Completion, but it's expected that the correct Requester will eventually timeout and take the appropriate action, so that error Message would be a low priority.

# Chapter 15: Error Detection and Handling

## Completion Timeout

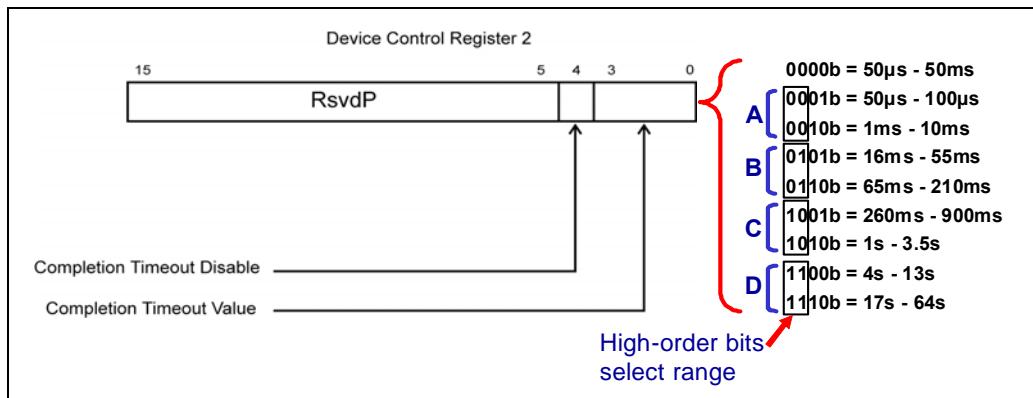
For the case of a pending Request that never receives the Completion it's expecting, the spec defines a Completion timeout mechanism. The spec clearly intends this to detect when a Completion has no reasonable chance of returning; it should be longer than any normal expected latencies.

The Completion timeout timer must be implemented by all devices that initiate Requests that expect Completions, except for devices that only initiate configuration transactions. Note also that every Request waiting for Completions is timed independently, and so there must be a way to track time for each outstanding transaction. The 1.x and 2.0 versions of the spec defined the permissible range of the timeout value as follows:

- It is strongly recommended that a device not timeout earlier than 10ms after sending a Request; however, if the device requires greater granularity a timeout can occur as early as 50 $\mu$ s.
- Devices must time-out no later than 50ms.

Beginning with the 2.1 spec revision, the Device Control Register 2 was added to the PCI Express Capability Block to allow software visibility and control of the timeout values, as shown in Figure 15-8 on page 665.

Figure 15-8: Device Control Register 2



If Requests need multiple Completions to return the requested data, a single Completion won't stop the timer. Instead, the timer continues to run until all the data has been returned regardless of how many Completions are needed. If only part of the data has been returned when the timeout occurs, the Requester may discard or keep that data.

## Link Flow Control Related Errors

Prior to forwarding the packet to the Data Link Layer for transmission, the Transaction Layer must check Flow Control (FC) credits to ensure that the receive buffers of the Link neighbor have sufficient room to hold it. Flow Control violations may occur, and they are considered uncorrectable. Protocol violations related to Flow Control can be detected by and associated with the port receiving the Flow Control information. Some examples are given here:

- Link partner fails to advertise at least the minimum number of FC credits defined by the spec during FC initialization for any Virtual Channel.
- Link partner advertises more than the allowed maximum number of FC credits (up to 2047 unused credits for data payload and 127 unused credits for headers).
- Receipt of FC updates containing non-zero values in credit fields that were initially advertised as infinite.
- A receive buffer overflow, resulting in lost data. This check is optional but a detected violation is considered to be a Fatal error.

---

## Malformed TLP

TLPs arriving in the Transaction Layer are checked for violations of the packet formatting rules. A violation in the packet format is considered a Fatal error because it means the transmitter has made a grievous mistake in protocol, such as failing to properly maintain its counters, and the result is that it's no longer performing as expected. Some examples of a packet being considered malformed (badly formed) include the following:

- Data payload exceeds Max payload size.
- Data length does not match length specified in the header.
- Memory start address and length combine to cause a transaction to cross a naturally-aligned 4KB boundary.
- TLP Digest (TD field) indication doesn't correspond with packet size (ECRC is unexpectedly missing or present).
- Byte Enable violation.
- Undefined Type field values.
- Completion that violates the Read Completion Boundary (RCB) value.
- Completion with status of Configuration Request Retry Status in response to a Request other than a configuration Request.
- Traffic Class field contains a value not assigned to an enabled Virtual Channel (this is also known as TC Filtering).

# Chapter 15: Error Detection and Handling

---

- I/O and Configuration Request violations (checking optional) - examples: TC field, Attr[1:0], and the AT field must all be zero, while the Length field must have a value of one.
- Interrupt emulation messages sent downstream (checking optional).
- TLP received with a TLP Prefix error:
  - TLP Prefix but no TLP Header
  - End-to-End TLP Prefixes preceding Local Prefixes
  - Local TLP Prefix type not supported
  - More than 4 End-to-End TLP Prefixes
  - More End-to-End TLP Prefixes than are supported
- Transaction type requiring use of TC0 has a different TC value:
  - I/O Read or Write Requests and corresponding Completions
  - Configuration Read or Write Requests and corresponding Completions
  - Error Messages
  - INTx messages
  - Power Management messages
  - Unlock messages
  - Slot Power messages
  - LTR messages
  - OBFF messages
- AtomicOp operand doesn't match an architected value.
- AtomicOp address isn't naturally aligned with operand size.
- Routing is incorrect for transaction type (e.g., transactions requiring routing to Root Complex detected moving away from Root Complex).

---

## Internal Errors

### The Problem

The first versions of the PCIe spec did not include a mechanism for reporting errors within a device that were unrelated to transactions on the interface itself. For Endpoints this wasn't really a problem because they have a vendor-specific device driver associated with them that can detect and report internal errors. However, Switches are considered system resources that are managed by the OS, and typically don't have software to help with internal error detection. In high-end systems, the ability to contain errors is important, so Switch vendors created proprietary means of handling internal errors. Unfortunately, since different vendor solutions were incompatible with each other, the end result was that they were seldom used.

## The Solution

To alleviate this situation, a standardized internal error reporting option was added with the 2.1 spec version. The definition of what constitutes an internal error is beyond the scope of the spec, but they can be reported as either Corrected or Uncorrectable Internal Errors.

A Corrected Internal Error means an error was masked or worked around by the hardware with no loss of information or improper behavior. An example would be an ECC error on an internal memory location that was corrected automatically. On the other hand, an Uncorrectable Internal Error means improper operation has resulted with potential data loss, such as a parity error on an internal memory location. Reporting internal errors is optional and, if it is used, the AER (Advanced Error Reporting) registers must be present to support it.

---

## How Errors are Reported

---

### Introduction

PCI Express includes three methods of reporting errors, as shown below. The first two, Completions and poisoned packets, were covered earlier, so our next topic will be the error Messages.

- Completions — Completion Status reports errors back to the Requester
- Poisoned Packet — reports bad data in a TLP to the receiver
- Error Message — reports errors to the host (software)

---

### Error Messages

PCIe eliminated the sideband signals from PCI and replaced them with Error Messages. These Messages provide information that could not be conveyed with the PERR# and SERR# signals, such as identifying the detecting Function and indicating the severity of the error. Figure 15-9 illustrates the Error Message format. Note that they're routed to the Root Complex for handling. The Message Code defines the type of Message being signaled. Not surprisingly, the spec defines three types of error Messages, as shown in Table 15-2.

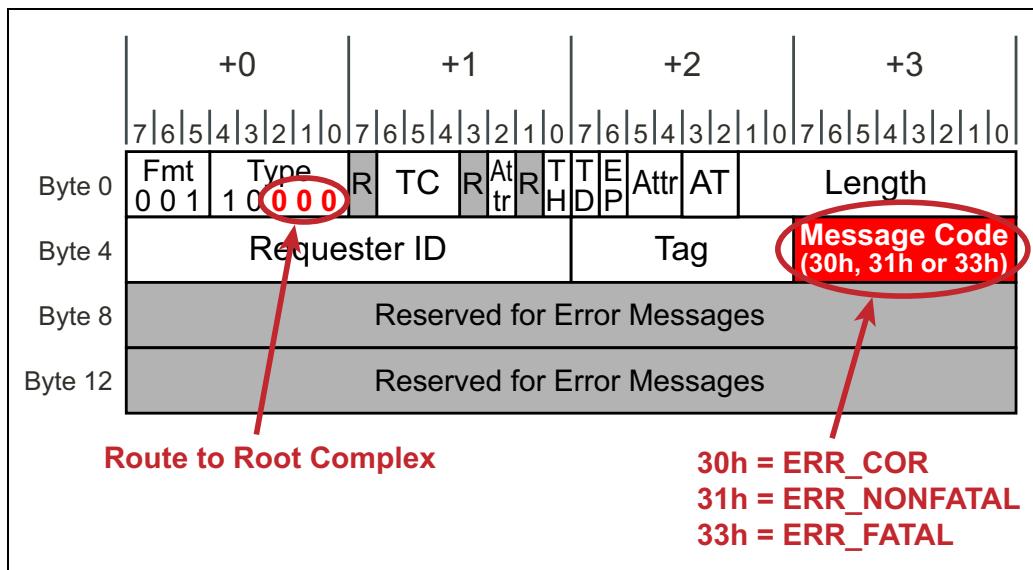
# Chapter 15: Error Detection and Handling

---

*Table 15-2: Error Message Codes and Description*

Message Code	Name	Description
30h	ERR_COR	Device detected a correctable error. This is automatically corrected by hardware and doesn't require software attention. However, it can be helpful to report them anyway so software can watch for trends like an increasing number of correctable errors.
31h	ERR_NONFATAL	Indicates an uncorrectable Non-Fatal error. No hardware correction mechanism was available but the Link is still working reliably. Software attention will be required to resolve the problem.
33h	ERR_FATAL	Indicates an uncorrectable Fatal error. No hardware correction mechanism was available and Link operation has failed in some important respect. Software attention will be required and a reset of at least one device will probably be required to resolve this issue.

*Figure 15-9: Error Message Format*



# PCI Express Technology

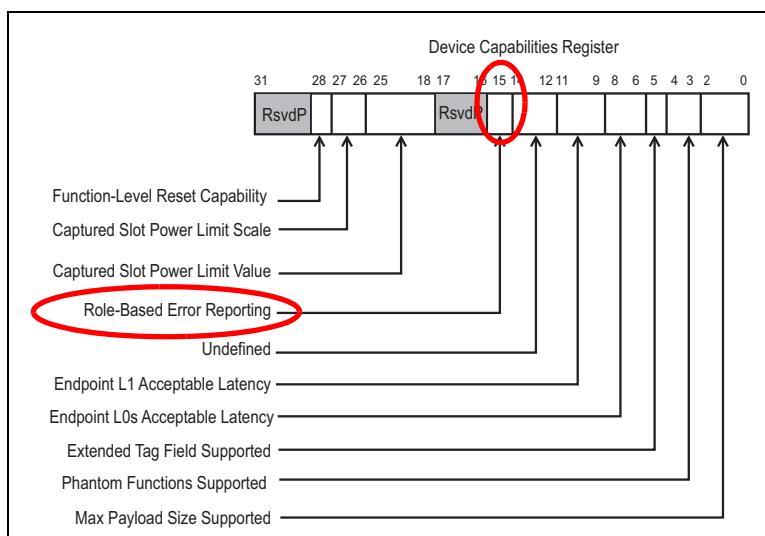
---

## Advisory Non-Fatal Errors

Since we've just seen that both types of Uncorrectable errors will need software attention, it sounds counter-intuitive to say that there are cases where it's preferable that a device not report Non-Fatal errors it detects, but there are. These cases are predominantly based on the role of the detecting agent (Requester, Completer, or Intermediate device) and the type of error. The problem is that multiple devices might report an error caused by the same event and, on some platforms, sending one of the Non-Fatal Error Messages (ERR\_NONFATAL) can prevent software from properly handling the error. For example, if an Endpoint reports an error, its device driver will be called to service the situation. However, if a Switch reports an error first for the same transaction, system software might be called to investigate and might not understand what the driver was trying to accomplish or what would be the optimal response.

That example illustrates that some detecting agents aren't the best ones to determine the ultimate disposition of the error and shouldn't send an uncorrectable message. Instead, such an agent can signal an advisory notification to software with ERR\_COR. This avoids confusion about the source of the uncorrectable error but still gives software a little more information about what happened. Eventually, the appropriate detecting agent will send the ERR\_NONFATAL message whenever it sees the error. Beginning with the 1.1 spec revision, a new field was added in the PCI Express Device Capabilities register to indicate support for this capability as shown in Figure 15-10 on page 670. This bit must be set for every agent that is compliant with the 1.1 spec or later.

Figure 15-10: Device Capabilities Register



# Chapter 15: Error Detection and Handling

---

In spite of the reasons just described, software might want to stop operation as soon as some advisory errors are seen by an intermediate device. Since newer devices will always perform role-based error reporting, an override mechanism is needed. To handle this case, software can escalate the severity of the advisory errors from Non-Fatal to Fatal in the AER (Advanced Error Reporting) registers. Since there is no “advisory fatal” case, the error will now be reported as a Fatal Error (ERR\_FATAL), if enabled, regardless of the role of the device.

## Advisory Non-Fatal Cases

The spec lists five situations for which an advisory message (ERR\_COR) is preferred over a ERR\_NONFATAL message. In each of these cases, the detecting agent will handle the error as an Advisory Non-Fatal Error. This means that a Non-Fatal condition will be handled by sending an ERR\_COR, assuming the agent has AER registers and has enabled ERR\_COR. If it doesn't have AER registers or ERR\_COR was not enabled, it sends no Error Message. The five cases are as follows:

1. Completer sent a Completion with UR or CA Status. The expectation in this case is that the Requester will have a mechanism to handle the error when it sees the offending Completion and will be the best agent to send whatever Error Messages are needed. A ERR\_NONFATAL message from the Completer would just be confusing, so it must be handled as Advisory Non-Fatal (ERR\_COR).

Curiously, there is no PCIe mechanism for the Requester to report that it received a Completion with this status. Instead, a design-specific method like an interrupt will be needed to get device driver attention. An important example of this happens when the Root Complex receives a Completion with UR or CA status in response to a Configuration Read Request. On some platforms the response is to return all 1's to software for this case, to support backward compatibility with PCI enumeration (configuration probing) software.

2. Intermediate device detected an error. This case comes up in systems that employ Switches because a detecting agent may not be the final destination for a TLP. As an example of this, consider Figure 15-11 on page 672, showing a poisoned packet delivered through an intermediate Switch. The TLP is seen as a Non-Fatal error by the Switch but it can only signal an ERR\_COR message instead (as long as it's enabled to do so).

To explore this concept a little more, why wouldn't we want the Switch to report ERR\_NONFATAL? One reason is seen by looking at error tracking in the AER registers. Figure 15-12 on page 672 shows the AER registers that track the Source ID (BDF of the sending device) of Error Messages coming into a Root Port and we can see that there's only one space available for

# PCI Express Technology

uncorrectable errors. If multiple uncorrectable errors are seen, that fact will be noted but only the first source ID will be saved since it is considered to be the probable cause of subsequent errors. It's important, therefore, that uncorrectable errors come from the most appropriate device to report them. It's worth noting that it's still helpful for intermediate devices to report ERR\_COR, because it allows software to determine where the error was first detected.

Figure 15-11: Role-Based Error Reporting Example

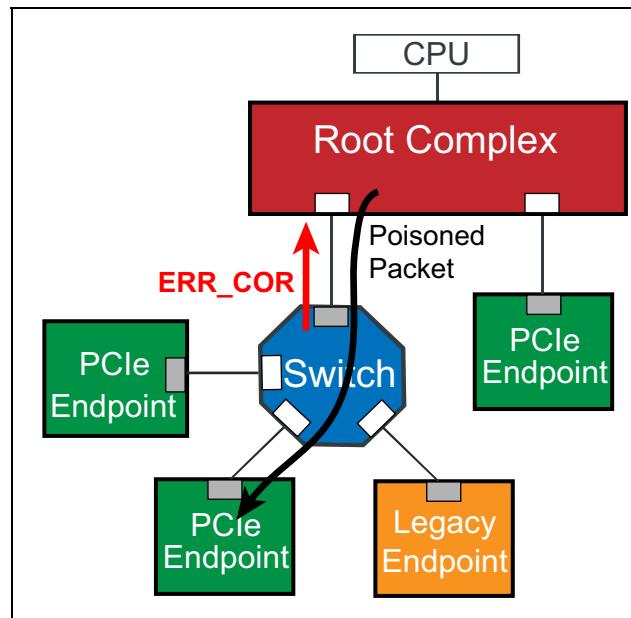
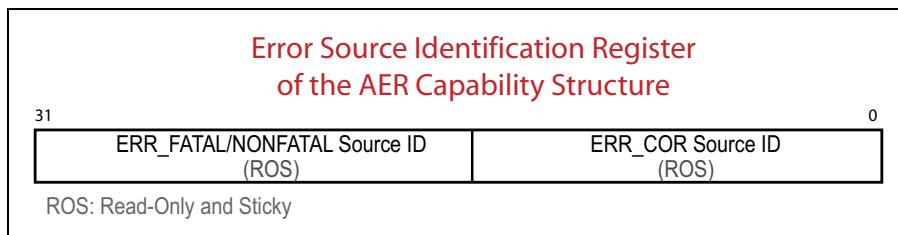


Figure 15-12: Advanced Source ID Register



## Chapter 15: Error Detection and Handling

---

As another example, 1.0a devices that have the UR Reporting Enable bit cleared but don't have the Role-Based Error Reporting capability are unable to report any error Messages when a UR error is detected (for posted or non-posted Requests). In contrast, a 1.1-compliant or later Completer that has the SERR# Enable bit set will send an ERR\_NONFATAL or ERR\_FATAL message for bad posted Requests, even if the Unsupported Request Reporting Enable bit is clear, so as to avoid silent data corruption. But it won't send an error Message for non-posted Requests received, so as to support the PCI-compatible configuration method of probing with configuration reads. It's recommended that software keep the UR Error Reporting Enable bit clear for devices that are not capable of Role-Based Error Reporting, but set it for those that are. That way, UR errors are reported on bad posted requests, but not for bad non-posted requests like configuration probing transactions, and backward compatibility with older software is maintained.

The spec also mentions that poisoned TLPs sent to the Root will be handled in the same way if the Root is acting as an intermediate agent, but there is one exception: If the Root doesn't support Error Forwarding, it will be unable to communicate the poisoned error with the TLP and must report this as a Non-Fatal error instead.

3. Destination device received a poisoned TLP. Normally, Endpoints would report the Non-Fatal error in this case, but there's an exception to this rule: If the ultimate destination device is able to handle the poisoned data in a way that allows for continued operation, it must treat this case as an Advisory Non-Fatal Error instead.

An example of this behavior might be an audio device that receives streaming data that has been poisoned. In this situation, the data may be accepted even though it's known to be corrupted because pausing the audio flow long enough to get software attention and take remedial action would be a worse alternative than allowing a glitch in the sound output.

4. Requester experienced a Completion Timeout. This is a similar case to the previous one; if the Requester has a means of continuing operation in spite of the problem then it must treat this as an Advisory Non-Fatal Error. A simple work-around for the Requester in this case would simply be to send the request again and hope for better results this time. Clearly, this would only make sense if the previous request did not cause any side effects, but Requesters are permitted to do this as often as they like (although the spec says the number of retries must be finite).
5. Unexpected completion received. This must be handled as an Advisory Non-Fatal Error. The reason is that it was probably caused by a mis-routed Completion and the original Requester will eventually report a Completion timeout. To allow that other Requester to attempt a retry of the failed

# PCI Express Technology

---

request, it's important that the one that sees the Unexpected Completion not send an Non-Fatal message.

---

## Baseline Error Detection and Handling

This section defines the required support for detecting and reporting PCI Express errors. Compliant devices must include:

- PCI-Compatible support — required to honor PCI-compatible error control and status fields for older software that has no awareness of PCI Express.
- PCI Express Error reporting — uses standard PCIe structures to for error control and status which can be used by newer software that does have knowledge of PCI Express.

---

## PCI-Compatible Error Reporting Mechanisms

### General

PCI Express errors are mapped into the original PCI configuration register bits for backward compatibility, allowing error status and control to be accessible to PCI-compliant software. To understand the features available from the PCI-compatible point of view, consider the error-related bits of the Command and Status registers located within the Configuration header. Some of the field definitions have been modified to reflect the related PCIe error conditions and reporting mechanisms. The PCI Express errors tracked by the PCI-compatible registers are:

- Transaction Poisoning/Error Forwarding (synonymous to data parity error in PCI)
- Completer Abort (CA) detected by a Completer (synonymous to Target Abort in PCI)
- Unsupported Request (UR) detected by a Completer (synonymous to Master Abort in PCI)

As mentioned earlier, the PCI mechanism for reporting errors is the assertion of PERR# (data parity errors) and SERR# (unrecoverable errors). The PCI Express mechanisms for reporting these events are the Completion Status values in Completions and Error Messages to the Root.

# Chapter 15: Error Detection and Handling

## Legacy Command and Status Registers

Figure 15-13 on page 675 illustrates the Command register and the location of the error-related fields. These bits are set to enable baseline error reporting under control of PCI-compatible software. Table 15-3 defines the specific effects of each bit.

Figure 15-13: Command Register in Configuration Header

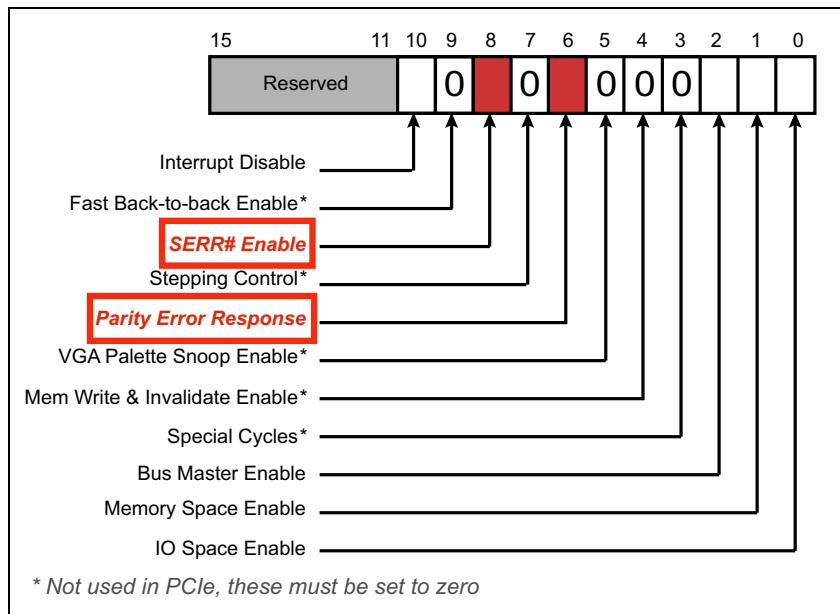


Table 15-3: Error-Related Fields in Command Register

Name	Description
SERR# Enable	Setting this bit enables sending ERR_FATAL and ERR_NONFATAL error messages to the Root Complex. These are considered roughly analogous to asserting the System Error (SERR#) signal in PCI. For Type 1 headers (bridges), this bit controls the forwarding of ERR_FATAL and ERR_NONFATAL error messages from the secondary interface to the primary interface. This field has no affect over ERR_COR messages.

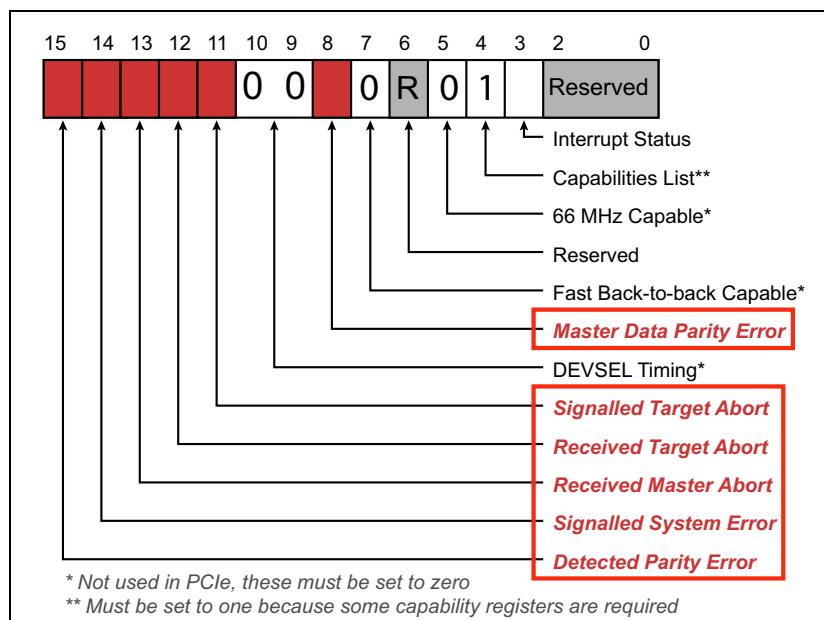
# PCI Express Technology

Table 15-3: Error-Related Fields in Command Register (Continued)

Name	Description
Parity Error Response	Setting this bit enables logging of poisoned TLPs in the Master Data Parity Error bit in the Status register. Poisoned packets indicate bad data and are roughly analogous to a PCI parity error.

Figure 15-14 on page 676 illustrates the Configuration Status register and the location of the error-related bit fields. Table 15-4 on page 677 defines the circumstances under which each bit is set and the actions taken by the device when error reporting is enabled.

Figure 15-14: Status Register in Configuration Header



# Chapter 15: Error Detection and Handling

---

Table 15-4: Error-Related Fields in Status Register

Error-Related Bit	Description
Detected Parity Error	Set by the port that receives a poisoned TLP. This status bit is updated regardless of the state of the Parity Error Response bit.
Signalled System Error	Set by a port that has reported an Uncorrectable Error with ERR_FATAL or ERR_NONFATAL and the SERR# enable bit in the Command register was set.
Received Master Abort	Set by a Requester that receives a Completion with status of UR (Unsupported Request). This is considered analogous to a PCI master abort because the target did not “claim the transaction”.
Received Target Abort	Set by a Requester that receives a Completion with status of CA (Completer Abort). This is analogous to a PCI target abort in that the target has had a programming violation or internal error condition.
Signaled Target Abort	Set by the Completer that handled a request (either posted or non-posted) as a Completer Abort. If it was a non-posted request, then a Completion with a Completion Status of CA is sent.
Master Data Parity Error	For Type 0 headers (e.g., Endpoints), this bit is set if the Parity Error Response bit in the Command register is set AND it either initiates a poisoned request OR receives a poisoned completion. For Type 1 headers (e.g., Switches and Root Ports), this bit is set if the Parity Error Response bit in the Command register is set AND it either initiates a poisoned request heading upstream OR receives a poisoned completion heading downstream.

---

## Baseline Error Handling

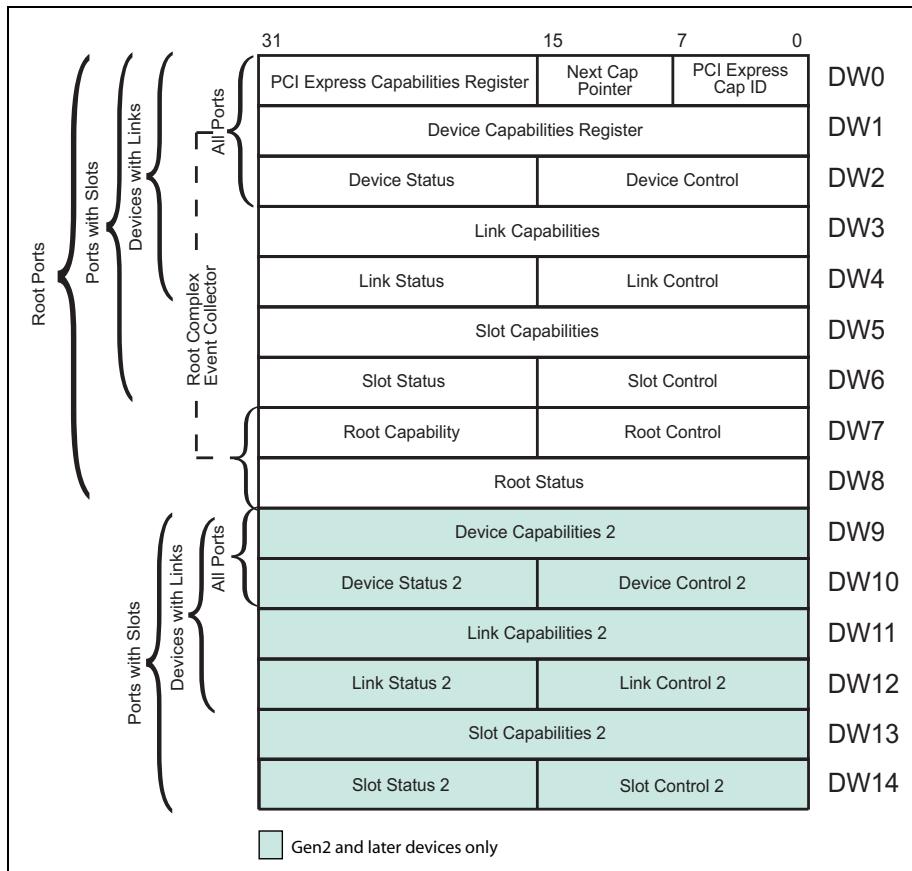
The Baseline capability requires the use of the PCI Express Capability structure. These registers include error detection and handling fields that provide finer granularity regarding the nature of an error and whether to report it or not than what is possible with just PCI-compatible error handling.

# PCI Express Technology

Figure 15-15 on page 678 illustrates the PCI Express Capability structure. Some of these registers provide support for:

- Enabling/disabling error reporting (Error Message Generation)
- Providing error status
- Providing link training status and initiating link re-training

Figure 15-15: PCI Express Capability Structure



## Enabling/Disabling Error Reporting

The Device Control registers allow software to enable generation of three different Error Messages for four error events, and Device Status registers allow it to see which error has been detected. The four error cases are:

# Chapter 15: Error Detection and Handling

---

- Correctable Errors
- Non-Fatal Errors
- Fatal Errors
- Unsupported Request Errors

Note that the only specific error identified here is the Unsupported Request. Although an Unsupported Request is technically a subset of Non-Fatal errors, and, when reported, is even signaled with an ERR\_NONFATAL message, it has its own enable and status bits. That's because during system enumeration Unsupported Requests are going to happen (whenever an attempt is made to read config space from a Function that doesn't actually exist in the system) but they must not be reported as errors. The enumeration software may have very limited error-handling capability and if it was required to stop and service an error it might fail. Therefore, the software doesn't want error messages generated for the UR case during that time, but does want to know about any other Non-Fatal errors that may be detected. (See the section titled "Discovering the Presence or Absence of a Function" on page 105 for more details on Unsupported Requests during enumeration.)

Table 15-5 on page 679 lists each error type and its associated error classification.

*Table 15-5: Default Classification of Errors*

Classification & Severity	Name of Error	Layer Detected
Correctable	Receiver Error	Physical
Correctable	Bad TLP	Link
Correctable	Bad DLLP	Link
Correctable	Replay Number Rollover	Link
Correctable	Replay Timer Timeout	Link
Correctable	Advisory Non-Fatal Error	Transaction
Correctable	Corrected Internal Error	
Correctable	Header Log Overflow	Transaction
Uncorrectable - Non Fatal	Poisoned TLP Received	Transaction
Uncorrectable - Non Fatal	ECRC Check Failed	Transaction

# PCI Express Technology

---

Table 15-5: Default Classification of Errors (Continued)

Classification & Severity	Name of Error	Layer Detected
Uncorrectable - Non Fatal	Unsupported Request	Transaction
Uncorrectable - Non Fatal	Completion Timeout	Transaction
Uncorrectable - Non Fatal	Completer Abort	Transaction
Uncorrectable - Non Fatal	Unexpected Completion	Transaction
Uncorrectable - Non Fatal	ACS Violation	Transaction
Uncorrectable - Non Fatal	MC Blocked TLP	Transaction
Uncorrectable - Non Fatal	AtomicOps Egress Blocked	Transaction
Uncorrectable - Non Fatal	TLP Prefix Blocked	Transaction
Uncorrectable - Fatal	Uncorrectable Internal Error (optional)	
Uncorrectable - Fatal	Surprise Down (optional)	Link
Uncorrectable - Fatal	Receiver Overflow (optional)	Transaction
Uncorrectable - Fatal	DLL Protocol Error	Link
Uncorrectable - Fatal	Receiver Overflow	Transaction
Uncorrectable - Fatal	Flow Control Protocol Error	Transaction
Uncorrectable - Fatal	Malformed TLP	Transaction

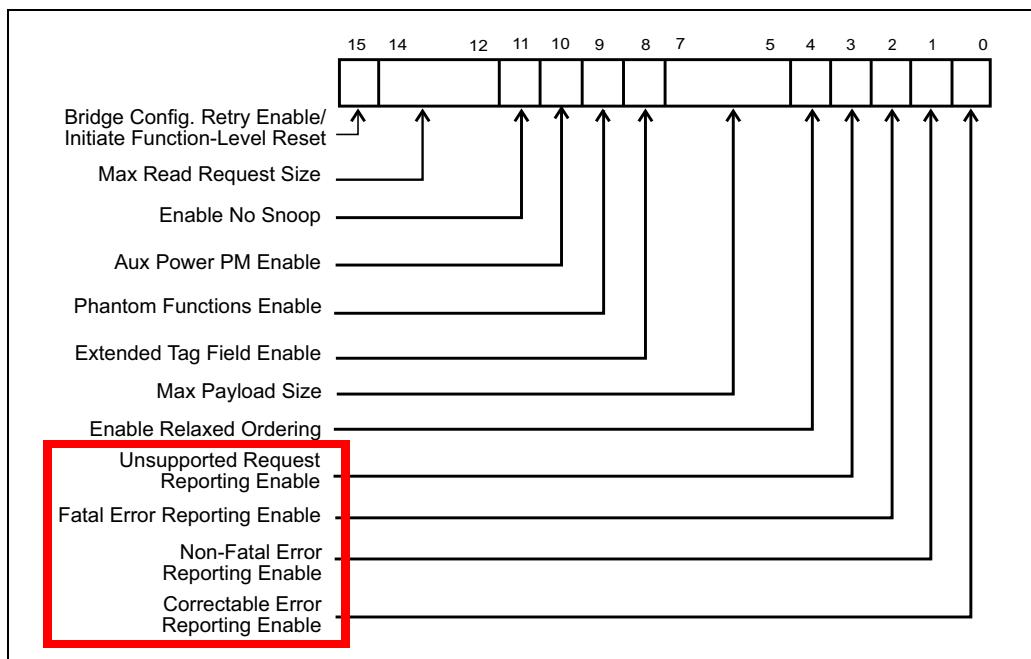
**Device Control Register.** Setting bits in the Device Control Register, shown in Figure 15-16 on page 681, enables sending the corresponding Error Messages to report errors. Unsupported Request errors are specified as Non-Fatal errors and are reported via a Non-Fatal Error Message, but only when the *UR Reporting Enable* bit is set.

In order for a Function to actually send an error message, either the corresponding enable bit in the Device Control register needs to be set, or for Fatal and Non-Fatal errors, the SERR# Enable should be set. For Uncorrectable Errors, if either the SERR# Enable bit in the Command Register is set OR the corresponding enable bit in the Device Control register is set, the appropriate error message will be sent (ERR\_FATAL or ERR\_NONFATAL).

## Chapter 15: Error Detection and Handling

For Correctable Errors, a Function will only send the ERR\_COR message if the *Correctable Error Reporting Enable* bit in the Device Control register is set. There is no control to enable ERR\_COR messages from the PCI-Compatible mechanisms, which makes sense because in PCI, there was no concept of correctable errors.

Figure 15-16: Device Control Register Fields Related to Error Handling

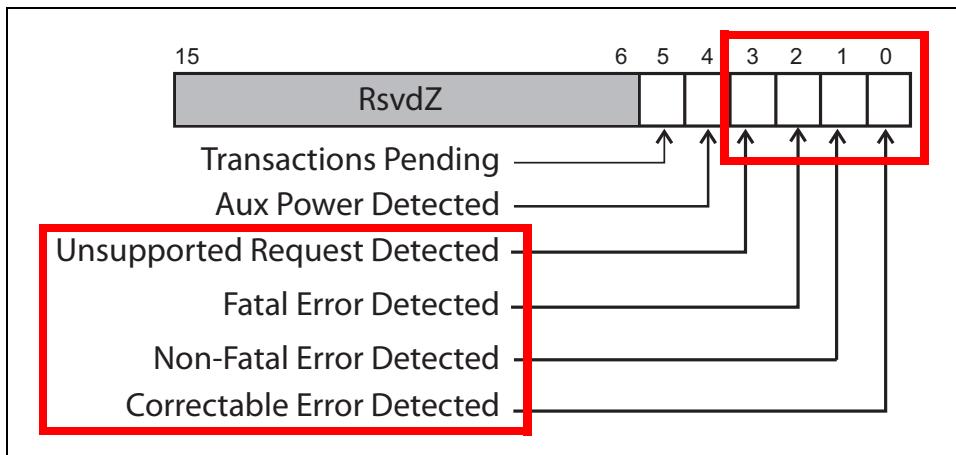


**Device Status Register.** An error status bit is set in the Device Status register, shown in Figure 15-17 on page 682, anytime an error associated with its classification is detected, regardless of the setting of the error reporting enable bits in the Device Control Register. Because Unsupported Request errors are considered Non-Fatal Errors, when these errors occur both the *Non-Fatal Error Detected* status bit and the *Unsupported Request Detected* status bit will be set. Like several other status bits, these are “Sticky” (their values are not cleared by a reset event so they’ll be available for diagnosing problems even if a reset was needed to get the Link working well enough to read the status).

# PCI Express Technology

---

Figure 15-17: Device Status Register Bit Fields Related to Error Handling



## Root's Response to Error Message

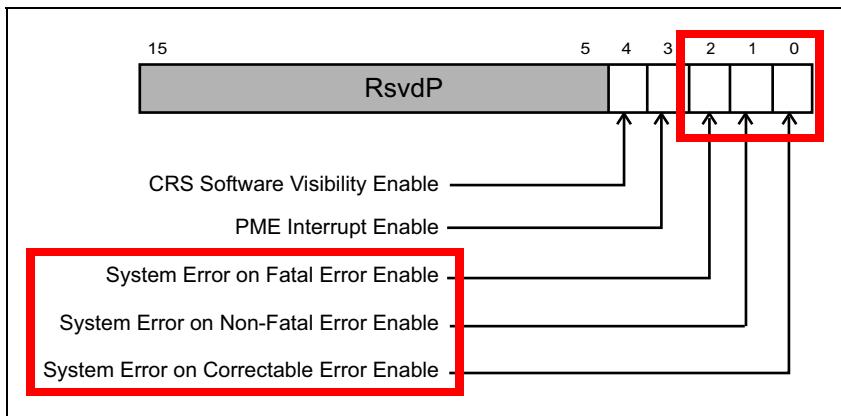
When an Error Message is received by the Root, the action it takes is determined in part by the settings in the Root Control Register. Figure 15-18 depicts this register and highlights the three fields that specify whether a received Error Message should be reported as System Error. In some x86-based systems, it's likely that an NMI (Non-Maskable Interrupt) will be signaled if the error is enabled to trigger a System Error.

Other options for reporting Error Messages are not configurable via standard registers. The most likely scenario is that an interrupt will be signaled to the processor that will call an Error Handler, which may log the error and attempt to clear the problem.

# Chapter 15: Error Detection and Handling

---

Figure 15-18: Root Control Register



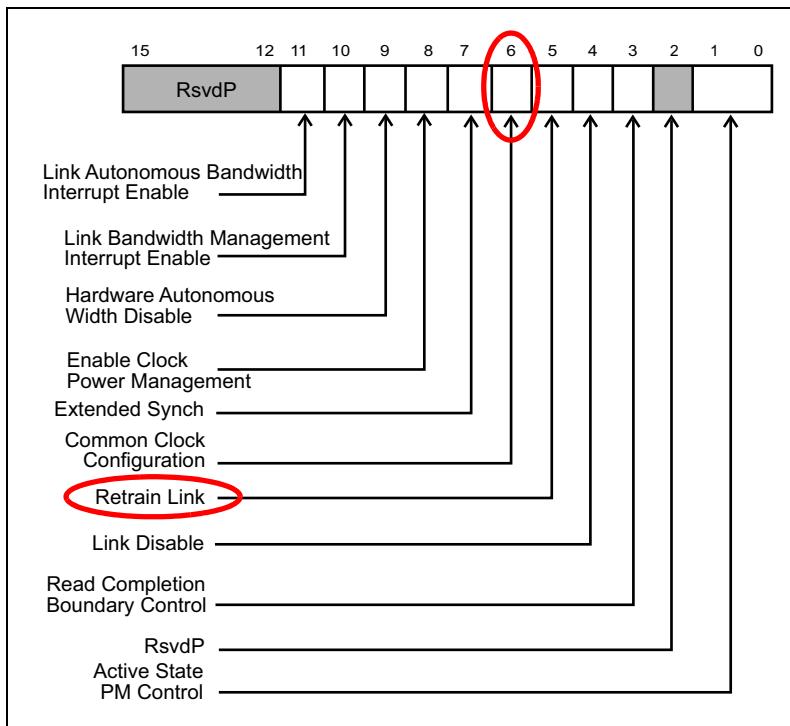
## Link Errors

Link failures are typically detected in the Physical Layer and communicated to the Data Link Layer. For a downstream device, if the link has incurred a Fatal error and is not operating correctly, it can't report the error to the host. For these cases, the error must be reported by the upstream device. If software can isolate errors to a given link, one step in handling an uncorrectable error (or to prevent future uncorrectable errors) is to retrain the Link. The Link Control Register includes a bit that allows software to force the Link to retrain, as shown in Figure 15-19 on page 684. If that solves the problem, operation resumes with little downtime.

# PCI Express Technology

---

Figure 15-19: Link Control Register - Force Link Retraining

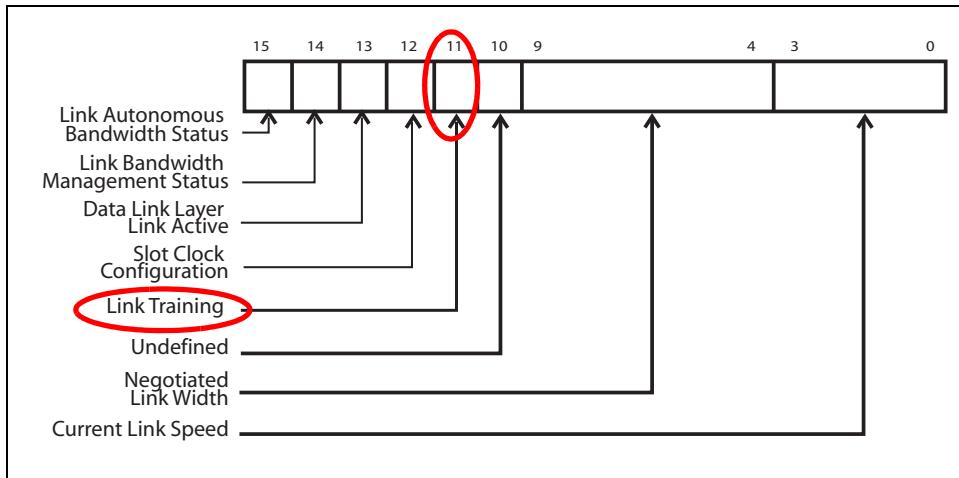


Having once requested retraining, software can poll the *Link Training* bit in the Link Status Register to see when training has completed. Figure 15-20 highlights this status bits. When this bit is 1b, the Link is still in the retraining process (or has yet to start retraining). Hardware will clear this bit once the Physical Layer reports the Link as active meaning the training process has completed successfully.

# Chapter 15: Error Detection and Handling

---

Figure 15-20: Link Training Status in the Link Status Register



---

## Advanced Error Reporting (AER)

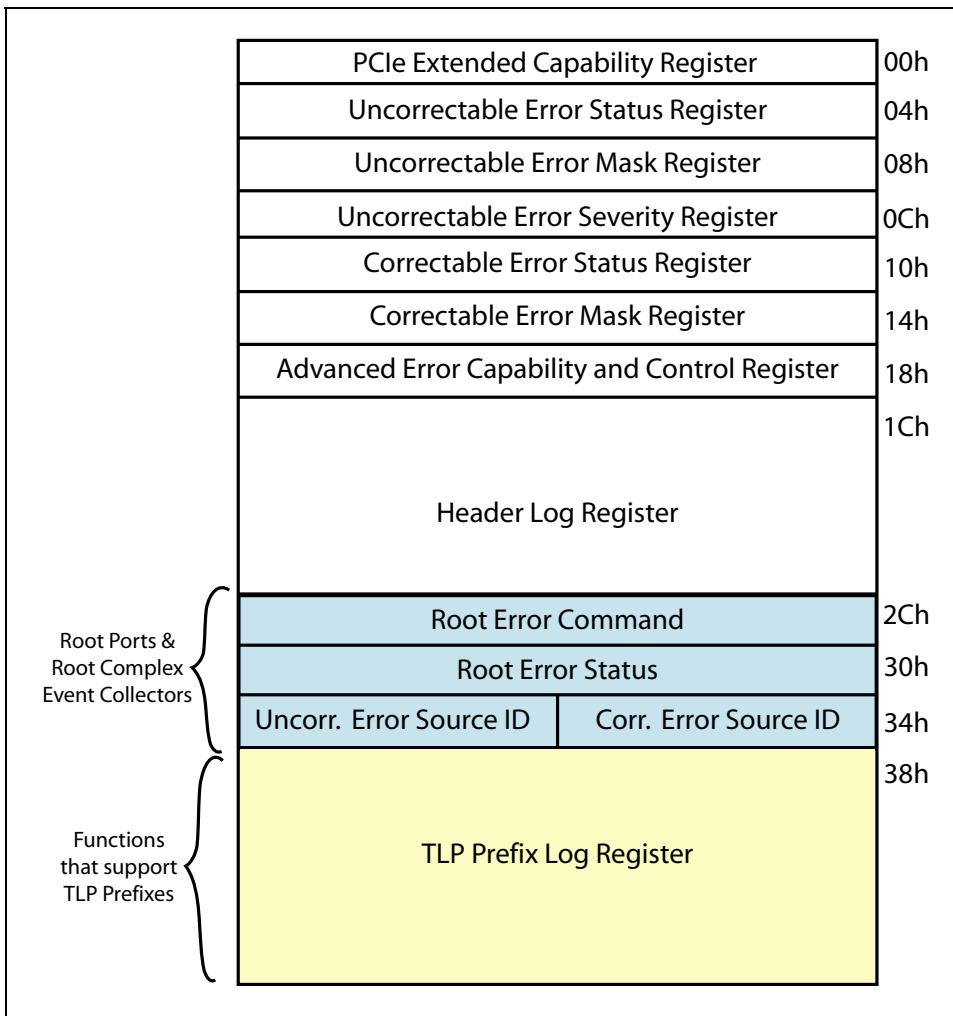
The Advanced Error Reporting Structure illustrated in Figure 15-21 on page 686 allows for much more sophisticated error handling. These registers provide several additional features:

- Better granularity in logging the actual type of error that occurred
- Control to specify the severity of each uncorrectable error type
- Support for logging the header of packets that had errors
- Standardizing control for the Root to report received Error Messages with an interrupt
- Identifying the source of the error in the PCIe topology
- Ability to mask reporting individual types of errors

# PCI Express Technology

---

Figure 15-21: Advanced Error Capability Structure



---

## Advanced Error Capability and Control

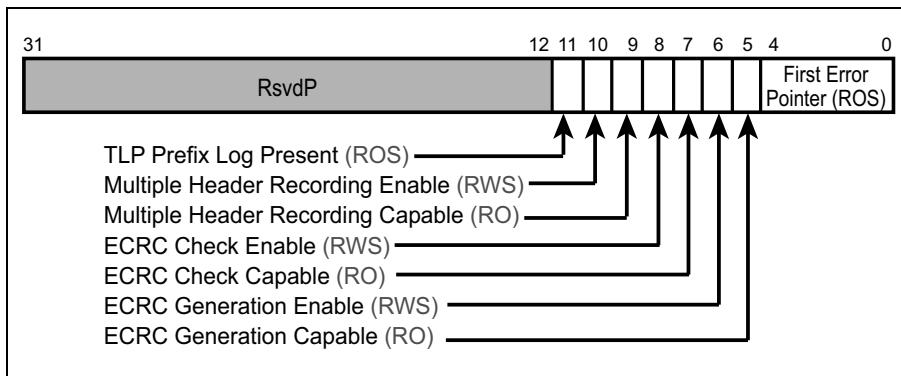
Let's begin our discussion of AER by looking at the Advanced Error Capability and Control register. End-to-End CRC (ECRC) generation and checking requires AER, and this register, shown in Figure 15-22 on page 687, reports

## Chapter 15: Error Detection and Handling

whether this device supports it. If so, configuration software can enable (and force) its use by setting the appropriate bits.

The five low-order bits of this register contain the First Error Pointer, set by hardware when the Uncorrectable Error status bits are updated. There are 32 status bits and the First Error Pointer indicates which of the unmasked, Uncorrectable Errors was detected first, meaning which status bit was set when all the other status bits were still 0. The first error is the most interesting because the others may have been caused by the first one.

Figure 15-22: The Advanced Error Capability and Control Register



Beginning with the 2.1 spec revision, this capability was enhanced to allow tracking multiple errors. For that reason, if multiple error status bits have been set and cleared, the meaning really becomes more like an “Oldest Error Pointer” instead. The pointer is updated by hardware when the corresponding status bit is cleared by software, at which time it points to whichever error was detected next (see Figure 15-25 on page 691 for the list of uncorrectable errors). Interestingly, the next error may be the same one again if that error had been detected multiple times, with the result that the updated pointer still indicates the same value.

Since multiple errors can be recorded in the Uncorrectable Status register, it would be very helpful to store multiple headers, too. Hardware must be designed to log at least one header, but is allowed to support more. If it does, the Multiple Header Recording Capable bit will be set and the Multiple Header Recording Enable bit can be used to enable storing more than one. Whenever the First Error Pointer indicates a status bit position that is not set or is not implemented, it means there are no more uncorrectable errors to service.

# PCI Express Technology

---

The last bit in this register, TLP Prefix Log Present, indicates whether the TLP Prefix Log registers contain valid information for the uncorrectable error indicated by the First Error Pointer.

The fields in this register and the other AER registers have various characteristics, which are abbreviated as follows:

- RO — Read Only, set by hardware
- ROS — Read Only and Sticky (see the next section on sticky bits)
- RsvdP - Reserved and Preserved. These bits must not be used for any purpose, but software must be careful to maintain whatever values they contain.
- RsvdZ - Reserved and Zero. Bits that must not be used for any purpose and must always be written to zeros.
- RWS — Readable, Writeable and Sticky
- RW1CS — Readable, Write 1 to Clear, and Sticky

---

## Handling Sticky Bits

Several AER register fields employ sticky bits, which means that a reset won't clear their contents. All other register fields are forced to default values on a reset, but these are not. This is a good idea because a Link may encounter a failure that can't be cleared without a reset. If the problem is in the downstream device of the failed Link, its register contents are unavailable until the Link is working again, which the reset will accomplish. But if the registers were cleared by the reset then the information is lost. To solve this problem, sticky bits keep error status information available through a reset. Specifically, sticky bits will survive an FLR (Function Level Reset), a Hot Reset, and a Warm Reset because power is available to keep them active. They may even survive a Cold Reset if a secondary power source like  $V_{aux}$  is available to keep them active when the main power is shut off.

---

## Advanced Correctable Error Handling

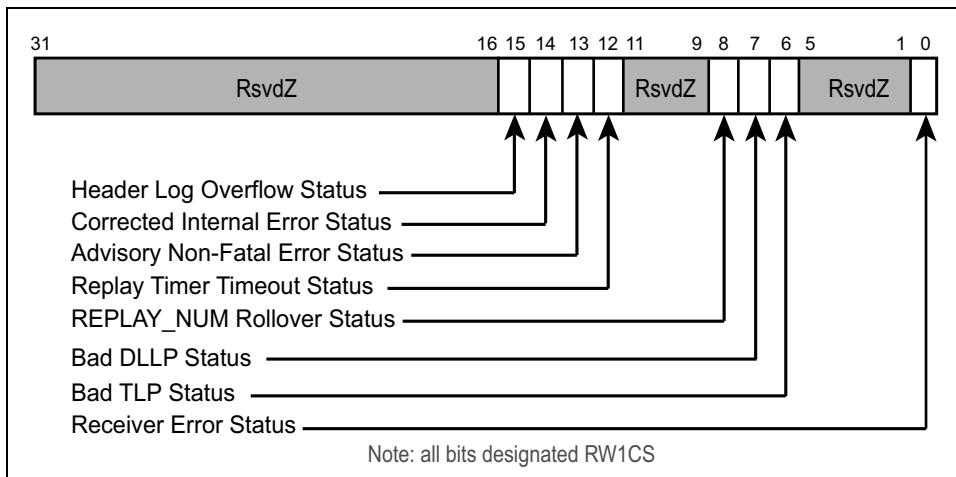
Advanced Error Reporting provides the ability to record which specific correctable errors have been detected. These errors can be used to initiate a Correctable Error Message to the host system. Although system operation continues normally, reporting correctable errors can be useful because it allows system software to see which components are having trouble and to predict whether they may fail completely in the future.

# Chapter 15: Error Detection and Handling

## Advanced Correctable Error Status

Correctable errors will automatically set the corresponding bit in the Advanced Correctable Error Status register, shown in Figure 15-23 on page 689, regardless of whether the error is reported with an Error Message. These bits are cleared by software writing a “1” to the bit position, hence the designation RW1CS.

Figure 15-23: Advanced Correctable Error Status Register



- Receiver Error (optional) — Physical Layer detected an error in the incoming packet. The packet is discarded at the Physical Layer, any buffer space allocated to it is released, and the Link Layer is informed that a receive error occurred.
- Bad TLP — Data Link Layer detected a packet with a bad LCRC, an out-of-sequence Sequence Number or an incorrectly nullified packet. In each case, the Link Layer discards the packet and reports a Nak DLLP to the transmitter, triggering a TLP replay.
- Bad DLLP — Data Link Layer noticed an incoming DLLP had a 16-bit CRC failure so the packet is dropped. A subsequent DLLP of the same type is expected to make up for the information it contained.
- REPLAY\_NUM Rollover — At the Data Link Layer, a set of TLPs have been sent without success (no Ack) four times in a row and this counter has rolled over back to zero. Hardware will automatically retrain the link in an attempt to clear the failure condition, then start the sequence again by replaying the contents of the Replay Buffer.

# PCI Express Technology

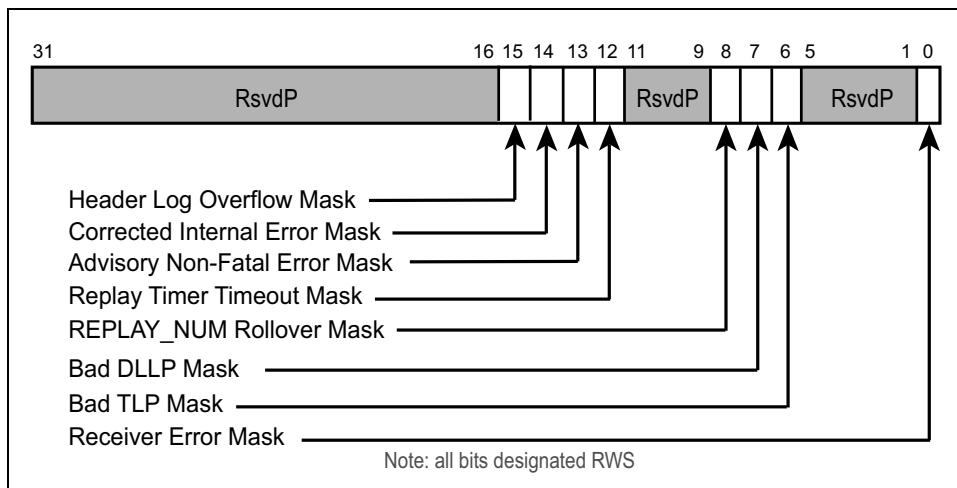
---

- Replay Timer Timeout — At the Data Link Layer, transmitted TLPs have not received an acknowledgement (Ack or Nak) within the timeout period. Hardware automatically replays all unacknowledged TLPs, meaning all packets in the Replay Buffer.
- Advisory Non-Fatal Error — Detection of these cases (see “Advisory Non-Fatal Errors” on page 670) is logged in the corresponding Uncorrectable Error Status register and as a correctable error here. It may also generate a Correctable Error Message, if enabled.
- Corrected Internal Error (optional) — An error internal to the device was detected, but it was corrected or worked around without causing improper behavior.
- Header Log Overflow (optional) — The maximum number of headers that can be stored in the header log has been reached. The number is just one if the Multiple Header Recording Enable bit is not set in the Advanced Error Capability and Control register.

## Advanced Correctable Error Masking

Correctable Error reporting is controlled collectively by the Correctable Error Enable bit in the Device Control register, but also individually by the Correctable Mask register, illustrated in Figure 15-24. The default state of the mask bits is cleared, meaning an ERR\_COR message can be delivered when any correctable errors are detected if they've been enabled (meaning the Correctable Error Enable bit is set). However, software may choose to set bits in this mask register to prevent a message from being sent when those specific errors are detected.

Figure 15-24: Advanced Correctable Error Mask Register



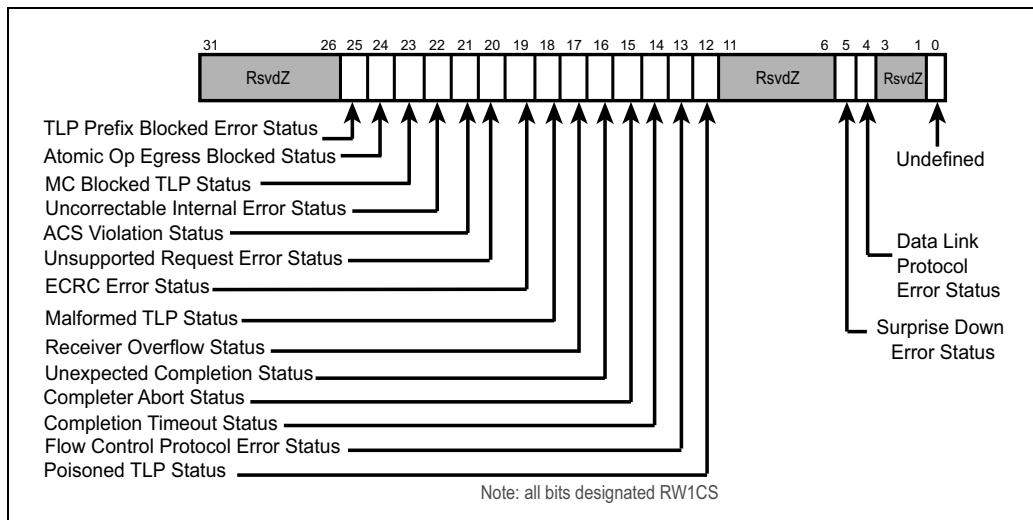
## Advanced Uncorrectable Error Handling

For uncorrectable errors, AER provides the ability to track which specific error has occurred, control whether it should be considered Fatal or Non-Fatal, and choose whether it will result in an Uncorrectable Error Message being sent to the Root.

### Advanced Uncorrectable Error Status

When an uncorrectable error occurs, the corresponding bit in this register is automatically set by hardware (see Figure 15-25 on page 691) regardless of whether the error will be reported to the Root. If multiple errors occur, hardware will set the corresponding bit for each error and will record which one was first in the First Error Pointer field of the Advanced Error Capability and Control register. It may even happen that multiple instances of the same error are detected before the first one can be serviced. Hardware that is compliant with the 2.1 spec revision or later will be able to keep track of a design-specific number of those cases.

Figure 15-25: Advanced Uncorrectable Error Status Register



The following list describes each of the register bits from right to left:

- Undefined — Previously, this first bit represented a link training failure at the Physical Layer, but that meaning was removed with the 1.1 revision of

the spec. Software must now ignore any value in this bit but may write any value to it. This information was no longer needed because bit 5, Surprise Down Error, now includes the same information in a broader meaning: the Link is not communicating at the Physical Layer.

- Data Link Protocol Errors — Caused by Data Link Layer protocol errors including the Ack/Nak retry mechanism. For example, a transmitter receives an Ack or Nak whose sequence number doesn't correspond to an unacknowledged TLP or to the ACKD\_SEQ number.
- Surprise Down — If the Physical Layer reports LinkUp = 0b (Link is no longer communicating) unexpectedly, this will be seen as an error unless it was an allowed exception. For example, if the Link Disable bit has already been set, then it's expected that LinkUp will be cleared and this condition won't be an error. This bit is only valid for Downstream Ports, which makes sense because it won't be possible to read status from an Upstream Port if the Link isn't working.
- Poisoned TLP — TLP was seen that had the EP bit set.
- Flow Control Protocol Error (optional) — Errors associated with failures of the Flow Control mechanism. Example: receiver reports more than 2047 data credits.
- Completion Timeout — A Completion is not received within the required amount of time after a non-posted request was sent.
- Completer Abort (optional) — Completer cannot fulfill a Request due to problems with the Request or failure of the Completer.
- Unexpected Completion — Requester receives a Completion that doesn't match any Requests that are awaiting a Completion.
- Receiver Overflow (optional) — More TLPs have arrived than the Receive Buffer had room to accept, resulting in an overflow error.
- Malformed TLP — Caused by errors associated with a received TLP header (see "Malformed TLP" on page 666).
- ECRC Error (optional) — Caused by an ECRC check failure at the Receiver.
- Unsupported Request Error — Completer does not support the Request. Request is correctly formed and had no other errors, but cannot be fulfilled by the Completer, perhaps because it's an invalid command for this device.
- ACS Violation — Access control error was seen in a received posted or non-posted request.
- Uncorrectable Internal Error — An internal error detected in the device could not be corrected or worked around by the hardware itself.
- MC Blocked TLP — A TLP designated for Multi-Cast routing was blocked. For example, an Egress Port can be programmed to block any MC hits that arrive with untranslated addresses (see "Routing Multicast TLPs" on page 896).
- AtomicOp Egress Blocked — Egress Ports of routing elements can be pro-

grammed to block AtomicOps from being forwarded to agents that shouldn't see them (see "AtomicOps" on page 897).

- TLP Prefix Blocked Error — Egress Ports of routing elements can be programmed not to forward TLPs containing End-to-End TLP Prefixes. If they then see one, they'll drop the TLP and report this error. For more on this, see "TPH (TLP Processing Hints)" on page 899.

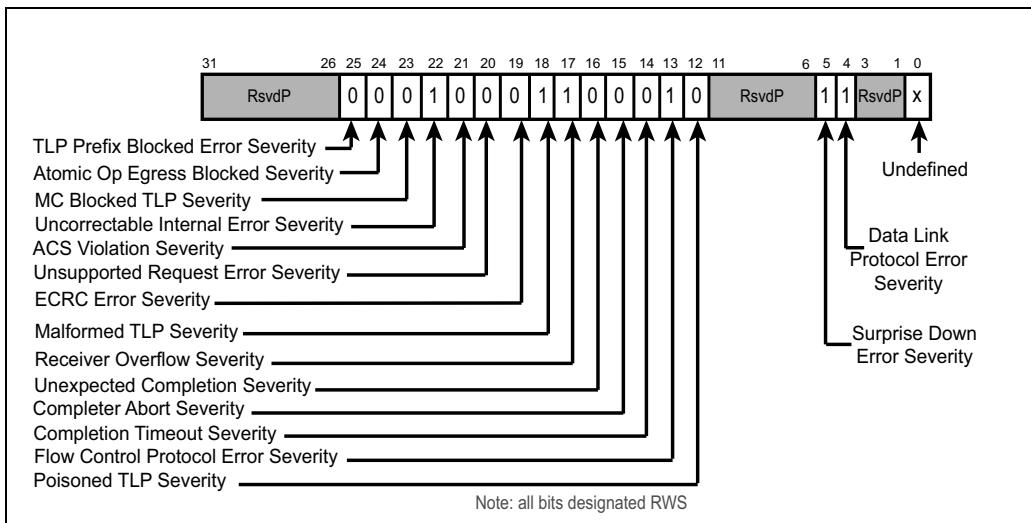
Recall that the First Error Pointer in the Capability and Control Register indicates which unmasked uncorrectable error was the first to arrive since the pointer was last updated. Error handling software can read the pointer to find out which error to investigate first. As an example, if the pointer value is 18d, that means bit position 18 in the Uncorrectable Status register was first, which is a Malformed TLP. Once that error has been serviced, software writes a one to bit 18 in the status register to clear that event, which updates the First Error Pointer to the next-most-recent error.

## Selecting Uncorrectable Error Severity

Software can select whether or not uncorrectable errors should be considered Fatal in this register, allowing errors to be treated differently for different applications. For example, a Poisoned TLP will be a Non-Fatal condition by default, and is treated as an Advisory Non-Fatal error in some cases, as discussed earlier. But software can escalate it to Fatal by setting its severity bit to one and then it will no longer be an advisory case. The default severity values are illustrated in the individual bit fields of Figure 15-26 on page 694 (1 = Fatal, 0 = Non-Fatal). If they are enabled and not masked, those errors selected as Non-Fatal will cause an ERR\_NONFATAL message to be sent to the Root Complex, and those selected as Fatal will cause an ERR\_FATAL message.

# PCI Express Technology

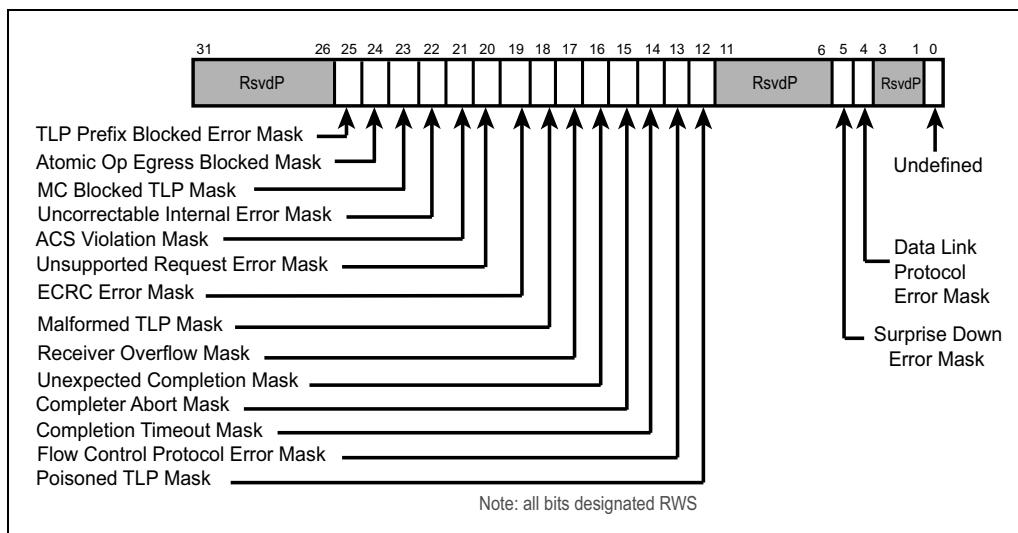
Figure 15-26: Advanced Uncorrectable Error Severity Register



## Uncorrectable Error Masking

Software can mask out individual errors so they won't cause an error message to be sent by using the Advanced Uncorrectable Error Mask register, shown in Figure 15-27 on page 694. The default condition is to allow Error Messages for each type of error (all mask bits are cleared).

Figure 15-27: Advanced Uncorrectable Error Mask Register



## Header Logging

A 4DW portion of the Advanced Error Reporting structure is used for storing the header of a received TLP that incurs an unmasked, uncorrectable error. Since header logging is only useful when a TLP has been received with a problem that wasn't seen by the Physical or Data Link Layers, the number of possibilities is limited, as shown in Table 15-6 on page 695. As mentioned earlier, when the optional AER capability is implemented, hardware is required to be able to log at least one header, though it may support logging more.

When the First Error Pointer is valid, the header log contains the header for the corresponding error if it was caused by an incoming TLP. Updating the Uncorrectable Error Status register will cause the Header Log registers to also update to the next value in sequence, meaning the next uncorrectable error that was detected. Since the hardware can only track a limited number of headers, it's important that software service uncorrectable errors quickly enough to avoid running out of header space. If the header log capacity is reached, that's a correctable error in itself (Header Log Overflow). This could happen if the number of supported log registers is exceeded or if the Multiple Header Log Enable bit is not set and the First Error Pointer is already valid when a new uncorrectable error is detected.

*Table 15-6: Errors That Can Use Header Log Registers*

Name of Error	Default Classification
Poisoned TLP Received	Uncorrectable - NonFatal
ECRC Check Failed	Uncorrectable - NonFatal
Unsupported Request	Uncorrectable - NonFatal
Completer Abort	Uncorrectable - NonFatal
Unexpected Completion	Uncorrectable - NonFatal
ACS Violation	Uncorrectable - NonFatal
Malformed TLP	Uncorrectable - Fatal

## Root Complex Error Tracking and Reporting

The Root Complex is the target of all error Messages from devices in a PCIe topology. Errors received by the Root update status registers and may be reported to the host system if enabled to do so.

### Root Complex Error Status Registers

When the Root receives an error Message, it sets status bits within the Root Error Status register (Figure 15-28 on page 697). This register indicates the type of error received and whether multiple errors of the same type have been received. Note that an error detected in the Root Port itself will set these status bits, too, as if the port had sent itself an error message. The status bits are:

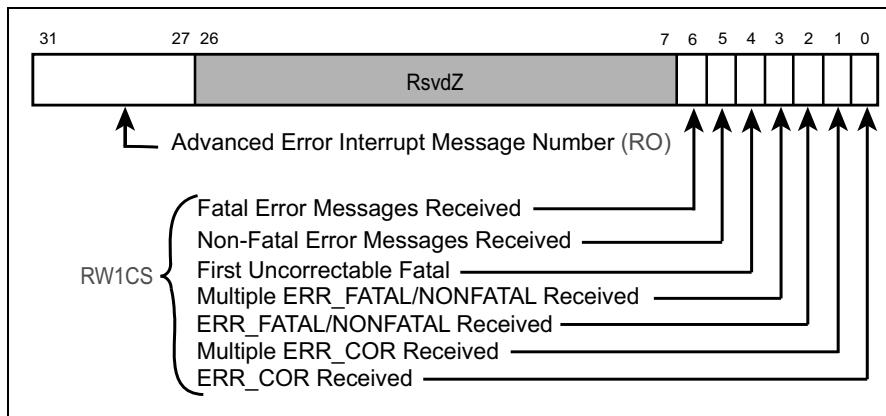
- ERR\_COR Received
- Multiple ERR\_COR Received - received an ERR\_COR message, or detected an unmasked Root Port correctable error with the ERR\_COR Received bit already set.
- ERR\_FATAL/NONFATAL Received
- Multiple ERR\_FATAL/NONFATAL Received - received an ERR\_FATAL or ERR\_NONFATAL message or detected an unmasked Root Port uncorrectable error with the ERR\_FATAL/NONFATAL Received bit already set.

It's possible for a system to implement separate software error handlers for Correctable, Non-Fatal, and Fatal errors, so this register includes bits to differentiate whether Uncorrectable errors were Fatal or Non-Fatal:

- If the first Uncorrectable Error Message received is Fatal the "First Uncorrectable Fatal" bit is also set along with the "Fatal Error Message Received" bit.
- If the first Uncorrectable Error Message received is Non-Fatal the "Non-fatal Error Message Received" bit is set. (If a subsequent Uncorrectable Error is Fatal, the "Fatal Error Message Received" bit will be set, but because the "First Uncorrectable Fatal" remains cleared, software knows that the first Uncorrectable Error was Non-Fatal).

# Chapter 15: Error Detection and Handling

Figure 15-28: Root Error Status Register

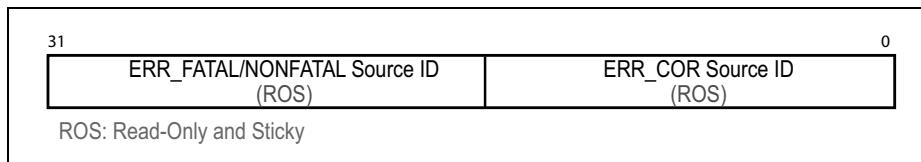


Finally, an interrupt may have been enabled (in the Root Error Command register) to be sent to the host system as a result of detecting one of these events. To support that, the 5-bit Interrupt Message Number in this register supplies the MSI or MSI-X vector number to be used, and there are 32 possibilities. For MSI, the number is the offset from the base data pattern. For MSI-X, it represents the table entry to be used, and must be one of the first 32 even if the agent supports more than 32. This read-only value is set by hardware and must be automatically updated if the number of MSI messages assigned to the device changes.

## Advanced Source ID Register

Software error handlers may need to read and clear status registers in the device that detected and reported the error. To facilitate this, the error Messages contain the ID (Bus:Dev:Func) of the first device reporting that error type. The Source ID register captures that ID from the Message for an incoming ERR\_FATAL/NONFATAL message if the ERR\_FATAL/NONFATAL bit isn't already set (meaning this is the first one). Similarly, the Source ID of the first received ERR\_COR message is captured, too, as shown in Figure 15-29 on page 698.

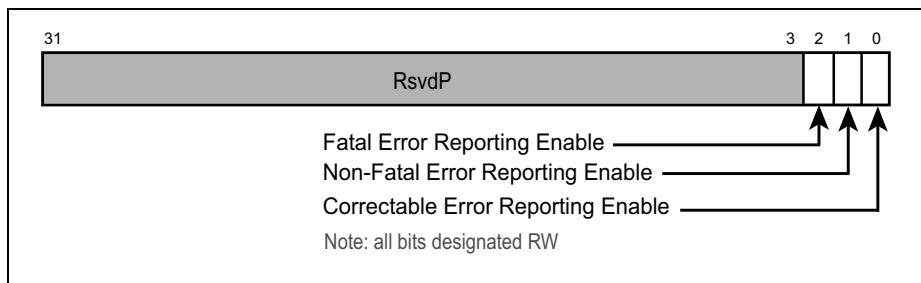
Figure 15-29: Advanced Source ID Register



## Root Error Command Register

The Root Complex has separate enable bits for each of the three error categories to control whether that error type will generate an interrupt to call an error handler as shown in Figure 15-30 on page 698. The interrupt that is generated will either be an MSI or MSI-X as discussed in “Root Complex Error Status Registers” on page 696. Once the interrupt is received, the called error handler would probably first read the Root Complex status registers to determine the nature of the error, and then go down to the source BDF of the error to read standard status register as well as possibly device-specific registers to determine what occurred and how it should be handled.

Figure 15-30: Advanced Root Error Command Register



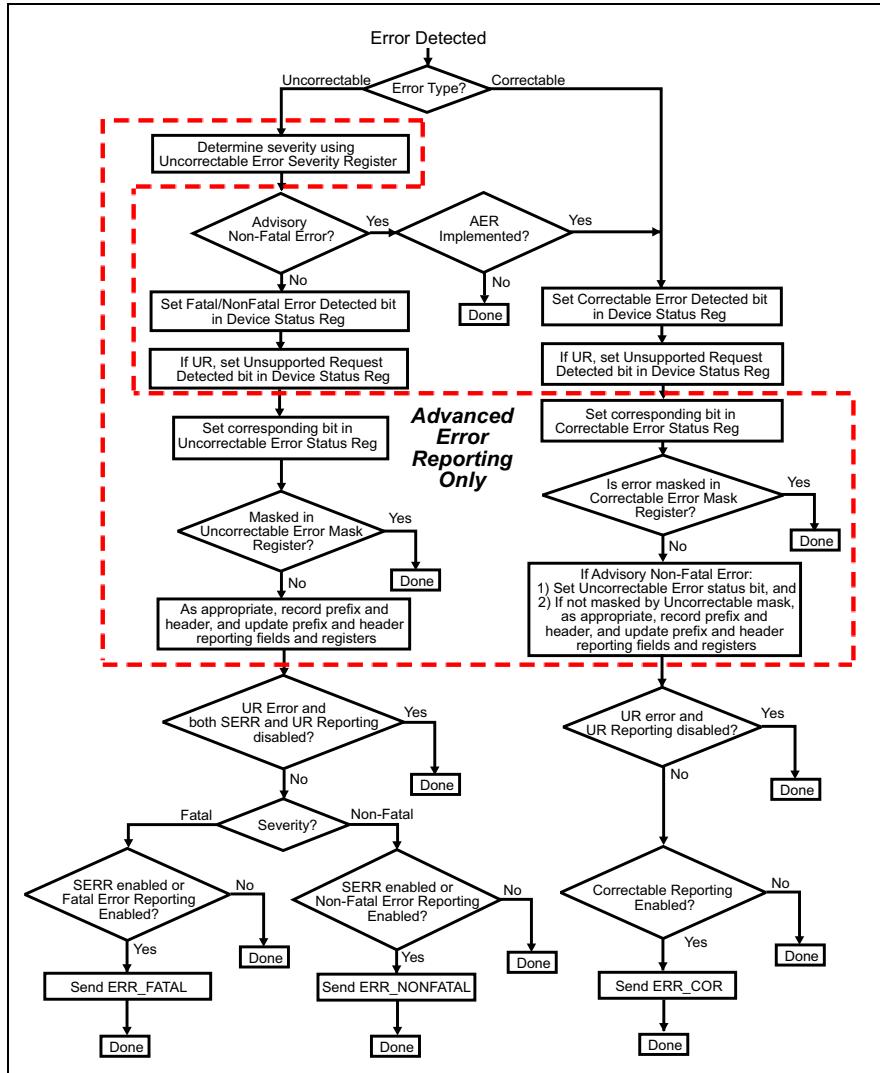
---

## Summary of Error Logging and Reporting

The spec includes the flow chart in Figure 15-31 on page 699 that shows the actions taken by a Function when an error is detected. The part inside the dashed line highlights the items that are added when the optional AER capability structure is present.

# Chapter 15: Error Detection and Handling

Figure 15-31: Flow Chart of Error Handling Within a Function



## Example Flow of Software Error Investigation

Now that we know all the mechanisms defined in PCIe for detecting, logging and reporting errors, it is worthwhile to look at how software would find and use this information to determine how to handle a reported error.

# PCI Express Technology

---

This example is going to assume that both the originating Function as well as the Root Port upstream of it both support AER. Without AER support, the standardized registers for error logging are very limited.

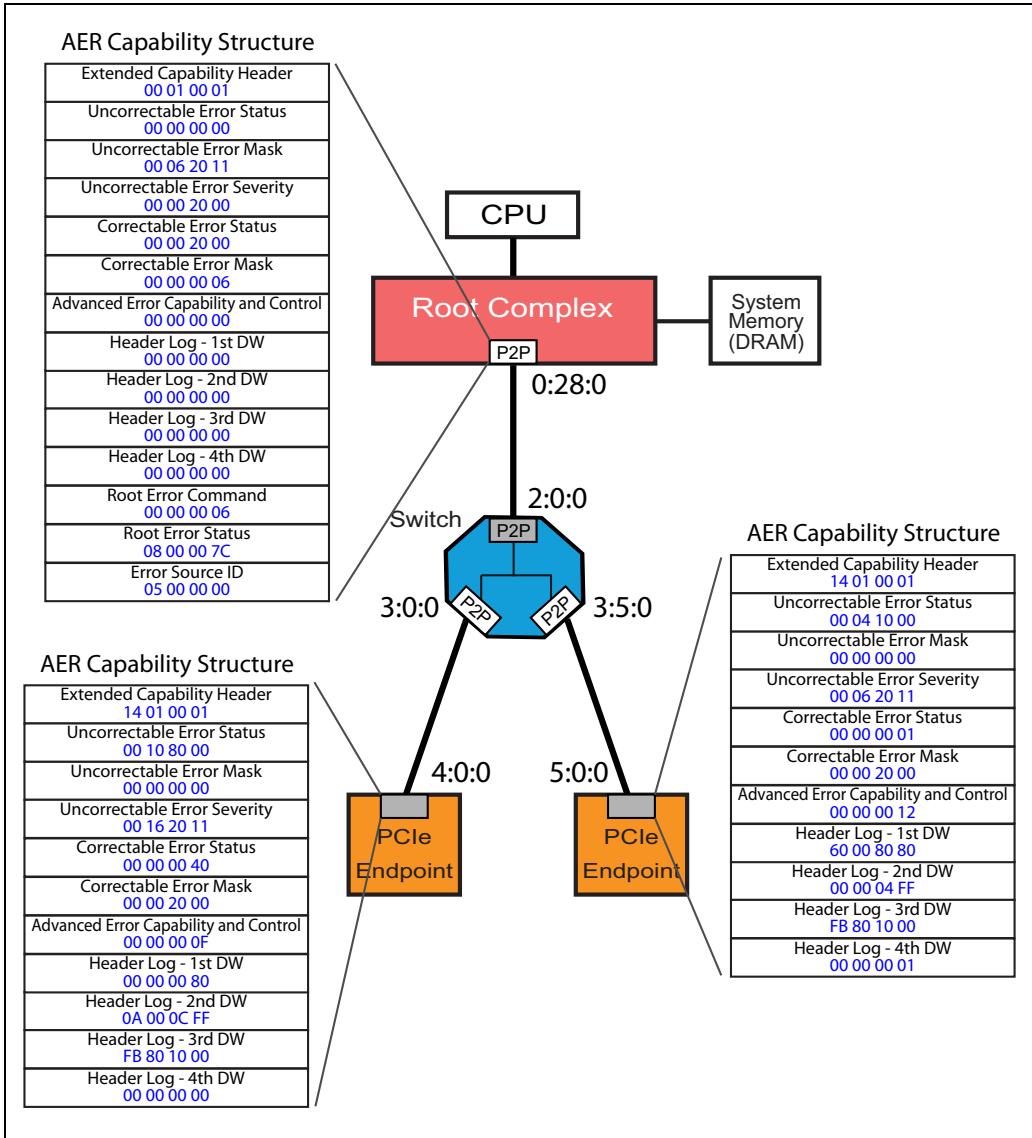
The system used for this example is shown in Figure 15-32 on page 701. The Root Port has a BDF of 0:28:0 and was enabled to generate an interrupt when it receives either an ERR\_FATAL or ERR\_NONFATAL message. We are going to follow the steps of error handling software would take to determine what errors have occurred, where they occurred and what packets were they detected in.

The error handling software has been called because of an interrupt from Root Port 0:28:0. The steps below are just an example, but illustrate the process of error handling software gathering error information.

1. Software knows it was Root Port 0:28:0 that called the error handler based on the interrupt vector used. Since MSI or MSI-X interrupts are used to report errors, each Root Port will have their own unique set of interrupt vectors.
2. The error handler reads the Root Error Status register of the AER structure on 0:28:0 to determine what types of error messages have been received by the Root Port. The value in that register is 0800\_007Ch which indicates that this Root Port has not received any ERR\_COR messages, but has received both ERR\_FATAL and ERR\_NONFATAL messages and the first uncorrectable error message that it received was an ERR\_FATAL.
3. The next step is to determine which BDF beneath this Root Port sent the first uncorrectable error. Software then reads the Source ID register of the Root Port and finds the value 0500\_0000h, which indicates that the source BDF of the first uncorrectable error was 5:0:0.
4. Now software knows that the first uncorrectable error received by Root Port 0:28:0 was a Fatal error that originated from BDF 5:0:0. With this information, software then goes and reads the Uncorrectable Error Status register on BDF 5:0:0 to see which specific uncorrectable errors have occurred on that BDF. The value returned from that read is 0004\_1000h which means that this BDF has detected at least one Malformed TLP and at least one Poisoned TLP. But what the error handler really cares about is which one occurred first, because that's the one that should be handled first.
5. To determine which of the multiple uncorrectable errors occurred first, software then reads the Advanced Error Capability and Control register of 5:0:0 and finds the value 0000\_0012h which has a First Error Pointer value of 12h meaning that the first uncorrectable error was a Malformed TLP (bit 18d) and not the Poisoned TLP (bit 12d).

# Chapter 15: Error Detection and Handling

Figure 15-32: Error Investigation Example System



# PCI Express Technology

---

6. Now that the error handler knows that the first uncorrectable error at 5:0:0 was a Malformed TLP, it can check the Header Log register to see the header of the packet that was malformed, since this is one of the errors where a header is recorded. In reading the Header Log register it finds these four doublewords:
  - 6000\_8080h - 1st DW
  - 0000\_04FFh - 2nd DW
  - FB80\_1000h - 3rd DW
  - 0000\_0001h - 4th DW
7. The evaluation of those 4 DWs identifies the malformed packet as: Memory Write, 4DW header, TC=0, TD=1, EP=0, Attr=0, AT=0, Length=80h (128 DWs or 512 bytes), Requester ID=0:0:0, Tag=4, Byte Enables=FFh, Address=1\_FB80\_1000h.  
The header of the packet all looks correct and every field uses valid encodings, so software must dig a little deeper to discover why this was treated as a Malformed TLP. In this example, let's assume that after further inspection of config space on 5:0:0, software discovers that the Max Payload Size enabled for this Function is 256 bytes, but this packet contained 512 bytes. This is a condition that will be treated as a Malformed TLP by the target device, in this case 5:0:0.

If you would like verify your knowledge of this error investigation process, go ahead and evaluate what the first uncorrectable error detected on 4:0:0 was.

If you're feeling adventurous and would like to check out this type of info on a real system, say your desktop or laptop, you can do so by downloading the MindShare Arbor software ([www.mindshare.com/arbor](http://www.mindshare.com/arbor)). You can run this on an x86-based machine and it will scan your system and display every visible PCI-compatible device with its configuration space decoded for easy interpretation.



---

# 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.

# PCI Express Technology

---

## 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.

# Chapter 16: Power Management

---

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.

# PCI Express Technology

---

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

Element	Responsibility
OS	<p>DIRECTS <b>OVERALL SYSTEM POWER MANAGEMENT</b> 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.</p>
ACPI Driver	<p>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).</p>
Device Driver	<p>The <b>Class driver</b> 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 <b>Miniport</b> driver supplied by the vendor of the specific device.</p> <p>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 <b>PCI Express Bus Driver</b> is the one to communicate with those registers.</p> <p>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.</p> <ul style="list-style-type: none"><li>• 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.</li><li>• 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.</li></ul>
Miniport Driver	<p><b>Supplied by the vendor of a device</b>, it receives requests from the Class driver and converts them into the proper series of accesses to the device's register set.</p>

# Chapter 16: Power Management

---

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

Element	Responsibility
PCI Express Bus Driver	This driver is <b>generic to all PCI Express-compliant devices</b> . It <b>manages their power states and configuration registers</b> , but does not have knowledge of a Function's device-specific register set (that knowledge is possessed by the Miniport Driver that the device driver uses to communicate with the device's register set). It receives requests from the device driver to change the state of the device's power management logic. For example: <ul style="list-style-type: none"><li>• When a request to power down the device is received, this driver is responsible for saving the context of the Function's PCI Express configuration registers. It then disables the ability of the device to act as a Requester or respond as a target and writes to the Function's PM registers to change its state.</li><li>• Conversely, when the device must be re-powered, the PCI Express Bus Driver writes to the PCI Express Function's PM registers to change its state and then restores the Function's configuration registers to their original state.</li></ul>
PCI Express PM registers within each Function's configuration space.	The <b>location, format and usage of these registers is defined by the PCIe spec</b> . The PCI Express Bus Driver understands this spec and therefore is the entity responsible for accessing a Function's PM registers when requested to do so by the Function's device driver.
System Board power plane and bus clock control logic	The implementation and control of this logic is typically system board design-specific and is therefore <b>controlled by the ACPI Driver</b> (under OS direction).

---

## ACPI Spec Defines Overall PM

The ACPI (Advanced Configuration and Power Interface) spec was first written several years ago as a joint effort by several companies to provide industry standards for OSPM (OS-level Power Management) in compute platforms. Power management at that time was being handled in proprietary ways on different platforms and that made it difficult for vendors to coordinate their efforts. In addition, platform-specific code wasn't always fully compatible with OS operations or aware of all the system conditions or policy considerations. ACPI helped in these areas by defining system power states, hardware registers and software interactions to accomplish OS-based power management. A detailed description of ACPI is beyond the scope of this book, but an introduction to the concepts and terminology will be helpful.

# PCI Express Technology

---

## System PM States

Table 16-2 on page 708 defines the possible states of the overall system with reference to power consumption. The “Working”, “Sleep”, and “Soft Off” states are defined in the OnNow Design Initiative documents.

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

Power State	Description
Working (G0/S0)	The system is fully operational.
Sleeping (G1)	<p>The system appears to be off and power consumption has been reduced. The amount of time it takes to return to the “Working” state is inversely proportional to the selected level of power conservation.</p> <ul style="list-style-type: none"><li>• S1 - caches flushed, CPU halted</li><li>• S2 - same as S1 except that now CPU is powered off. Not commonly used because it's not much better than S3.</li><li>• S3 - (also called “Suspend to RAM” or “Standby”) This is the same as S2 except that the system context is saved in memory and more of the system is shut down. When the system wakes up the CPU begins the full boot process but finds flags set in the CMOS memory that direct it to reload the context from RAM instead, and thus program execution can be resumed very quickly.</li><li>• S4 - (also called “Suspend to Disk” or “Hibernate”) Similar to S3, except that now the system copies the system context to disk, and then removes power from the system, including main memory. This gives better power savings but the restart time will be longer because the context must be restored from the disk before resuming program execution.</li></ul>
Soft Off (G2/S5)	The system appears to be off and power consumption is minimal. It requires a full reboot to return to the “Working” state because the contents of memory have been lost, but there is still some power available to do the wakeup, such as by pressing the “Power” button on the system.
Mechanical Off (G3)	The system has been disconnected from all power sources and no power is available.

# Chapter 16: Power Management

---

## Device PM States

ACPI also defines the PM states at the device level, which are listed in Table 16-3 on page 709. Table 16-3 on page 709 presents the same information in a slightly different form. The registers that support these device states must be implemented for PCIe devices.

Table 16-3: OnNow Definition of Device-Level PM States

State	Description
D0	<b>Mandatory.</b> Device is fully operational and uses full power from the system. The 2.1 spec revision added another set of registers to support 32 substates under D0 referred to as Dynamic Power Allocation registers.
D1	<b>Optional.</b> Low-power state in which device context may or may not be lost. No definition for this state is given, but it would represent a lower power state than D0 and higher than D2
D2	<b>Optional.</b> Presumably a lower power state than D1 that attains greater power savings, but would incur a longer recovery delay and may cause Device to lose some context.
D3	<b>Mandatory.</b> Device is prepared for loss of power and context may be lost whether the power actually goes off or not. Recovery time will be longer than for D2, but power can be removed from the device gracefully in this state.

## Definition of Device Context

**General.** During normal operation, the operational state of a Device is constantly changing. A device driver may write or read its registers, or a local processor on the Device may execute code that affects its interaction with the system. The state of the device at a given instant in time includes:

- The contents of its configuration registers.
- The state of its local memory and IO registers.
- If it contains a processor, then the current program pointer and contents of its other registers would be included.

This state information is referred to as the *device context*. Some or all of this may be lost if the Device PM state is changed to a more aggressive level. If the context information is not maintained, the Device won't operate correctly when it returns to the D0 (fully operational) state.

# PCI Express Technology

---

**PME Context.** If the OS enables a modem to wake the system for an incoming call and then powers down the system, the Device wake-up context will need to be retained locally during that time. The chipset retains enough power to allow it to monitor for these events. To support this feature, a PCIe modem must implement configuration registers including:

- PME Message capability.
- PME enable/disable control bit.
- PME status bit indicating whether the device has sent a PME message.
- One or more device-specific control bits that selectively enable or disable various device-specific events that can cause the device to send a PME message.
- Corresponding device-specific status bits that indicate why the device issued a PME message.

## Device-Class-Specific PM Specs

**Default Device Class Spec.** As mentioned earlier, ACPI gives four possible device power states (D0 - through - D3). It also defines the minimum PM states that all device types must implement, as listed in Table 16-4 on page 710.

*Table 16-4: Default Device Class PM States*

State	Description
D0	Device is on, is running at full power, and is fully operational.
D1	This optional state is only defined as being lower power than D0. It is not commonly used.
D2	This optional state is only defined as being lower power than D1. It is not commonly used.
D3	Device consumes the minimum possible power and main power may be turned off. The only requirement is that, while power is still on, the device must be able to service a configuration command to re-enter D0. Power can be removed from the device in this state, and the device will experience a hardware reset when power is restored.

# **Chapter 16: Power Management**

---

**Device Class-Specific PM Specs.** Above and beyond the power states mandated by the *Default Device Class Spec*, certain device classes may require the intermediate power states (D1 and/or D2) or exhibit certain common characteristics in a particular power state.

The rules associated with a particular device class are found in the *Device Class Power Management Specs* available on Microsoft's Hardware Developers' web site. For example, Device Class Power Management Specs exist for the following classes:

- Audio
- Communications
- Display
- Input
- Network
- PC Card
- Storage

## **Power Management Policy Owner**

A Device's PM policy owner is defined as the software module that makes decisions regarding the PM state of a device. In a Windows environment, the policy owner is the class-specific driver associated with devices of that class.

---

## **PCI Express Power Management vs. ACPI**

### **PCI Express Bus Driver Accesses PM Registers**

As indicated in Table 16-1 on page 706 and Figure 16-1 on page 712, the PCI Express Bus Driver understands the location, format and usage of the PM configuration registers. It's called when the OS needs to change the power state of a PCIe device or determine its status and capabilities. Other examples include:

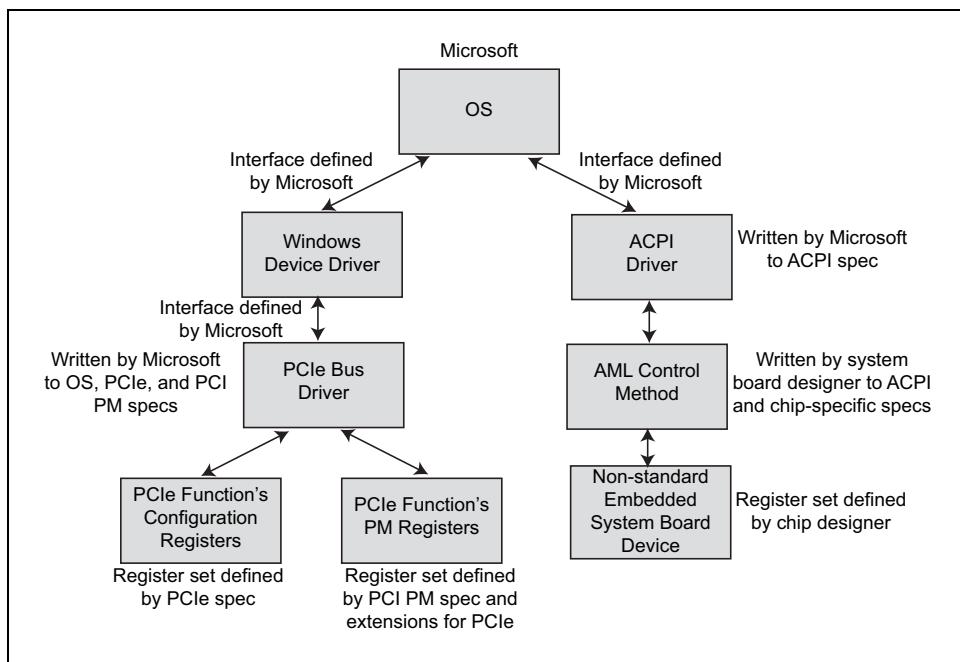
- The IEEE 1394 Bus Driver, which understands how to use the PM registers defined in the 1394 Power Management spec.
- The USB Bus Driver, which understands how to use the PM registers defined in the USB Power Management spec.

# PCI Express Technology

## ACPI Driver Controls Non-Standard Embedded Devices

There are devices embedded on the system board whose register sets do not adhere to any particular industry standard spec. At boot time, the BIOS reports these devices to the OS via the **ACPI tables**, also referred to as the **namespace**. When the OS needs to communicate with any of these devices, it calls the ACPI Driver, which executes a handler called a **Control Method** associated with the device. The handler is also found in the ACPI tables and is written by the platform designer using a special interpretive language called ACPI Source Language, or **ASL**. The ASL code is then compiled into ACPI Machine Language, or **AML**. Note that AML is not a processor-specific machine language. It's a tokenized (i.e., compressed) version of the ASL source code. An ACPI Driver incorporates an AML token interpreter that allows it to "execute" a Control Method.

Figure 16-1: Relationship of OS, Device Drivers, Bus Driver, PCI Express Registers, and ACPI



## Function Power Management

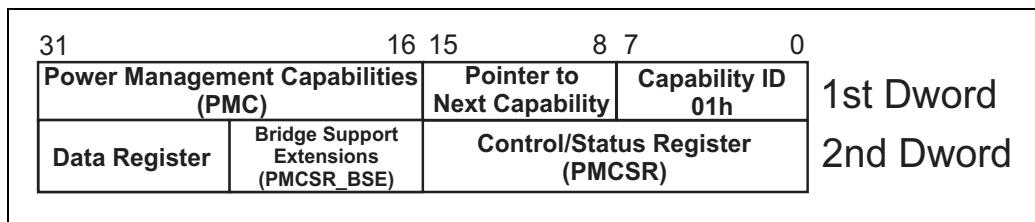
PCI Express Functions are required to support power management, and several registers and related bit fields must be implemented as discussed below.

### The PM Capability Register Set

The PCI-PM spec defines the Power Management Capability configuration registers. These registers were optional for PCI, but required for PCIe, and are located in the PCI-compatible configuration space with a Capability ID of 01h. Software can perform the following sequence to locate these registers:

1. Bit 4 of the Function's **Configuration Status register** should be set, indicating that the Capabilities Pointer in the first byte of dword 13d of the Function's configuration Header is valid. Reading the **Capabilities Pointer register** gives the offset to the first of the Function's linked list of capability registers.
2. If the least significant byte of the dword at that offset contains **Capability ID 01h** (see Figure 16-2 on page 713), this is the PM register set. The byte immediately following the Capability ID byte is the *Pointer to Next Capability* field that gives the offset in configuration space of the next Capability (if there is one). A non-zero value is a valid pointer, while a value of 00h indicates the end of the linked list. A description of all the PM registers can be found in "Detailed Description of PCI-PM Registers" on page 724.

Figure 16-2: PCI Power Management Capability Register Set



### Device PM States

Each PCI Express Function must support the full-on D0 state and the full-off D3 state, while D1 and D2 are optional. The sections that follow describe the possible PM states.

# PCI Express Technology

---

## D0 State—Full On

**Mandatory.** In this state, no power conservation is in effect and the device is fully operational. All PCIe Functions must support the D0 state and there are technically two substates: D0 Uninitialized and D0 Active. ASPM hardware control can change the Link power while the Device is in this state. Table 16-5 on page 714 summarizes the PM policies in the D0 state.

**D0 Uninitialized.** A Function enters D0 Uninitialized after a Fundamental Reset or, in some cases, when software transitions it from D3<sub>hot</sub> to D0. Usually, the registers are returned to their default state. In this state, the Function exhibits the following characteristics:

- It only responds to configuration transactions.
- Its Command register enable bits are all returned to their default states, meaning it cannot initiate transactions or act as the target of memory or IO transactions.

**D0 Active.** Once the Function has been configured and enabled by software, it is in the D0 Active state and is fully operational.

*Table 16-5: D0 Power Management Policies*

Link PM State	Function PM State	Registers or State that must be valid	Power	Actions permitted to Function	Actions permitted by Function
L0	D0 un-initialized	PME context **	< 10W	PCI Express config transactions.	None
L0 L0s (required)* L1 (optional)*	D0 active	all	full	Any PCI Express transaction.	Any transaction, interrupt, or PME. **
L2/L3	D0 active	N/A***			

\* Active State Power Management

\*\* If PME supported in this state.

\*\*\* This combination of Bus/Function PM states not allowed.

## Dynamic Power Allocation (DPA)

**Optional.** The 2.1 revision of the base spec added another optional capability that defines 32 more substates for D0 and describes their characteristics. This was intended to facilitate negotiation regarding power management between a

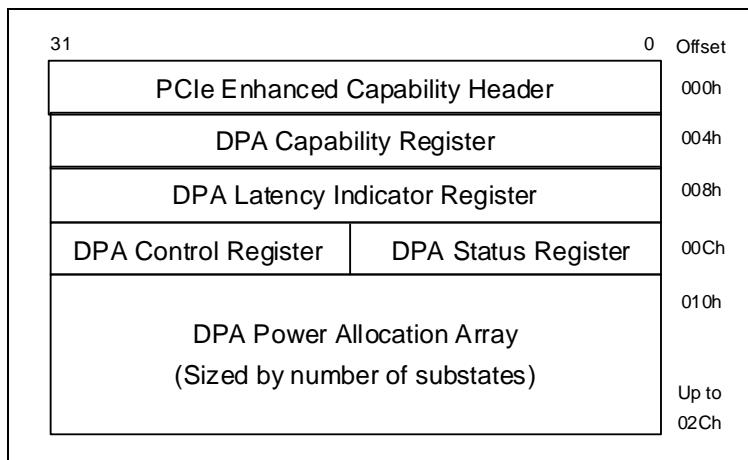
# Chapter 16: Power Management

---

device driver, OS, and an executing application, partly because some Functions don't have device drivers that handle PM well. One advantage of this model is that the Device technically still remains in the D0 state and may therefore be able to continue operating in a reduced capacity instead of going offline as would be caused by a change to the D1 or lower state.

DPA registers only apply when the Device power state is in D0 and aren't applicable in states D1-D3. Up to 32 substates can be defined, and they must be contiguously numbered from zero to the maximum value. Substate 0 is the initial default value and represents the maximum power the Function is capable of consuming. Software is not required to transition between substates in sequential order or even wait until a previous transition is completed before requesting another change in the substate. Consequently, when a Function has completed a substate change it must check the configured substate and, if they don't match, it must begin changing to the configured value. The registers to support DPA, illustrated in Figure 16-3 on page 715, are found in the Enhanced configuration space.

Figure 16-3: Dynamic Power Allocation Registers



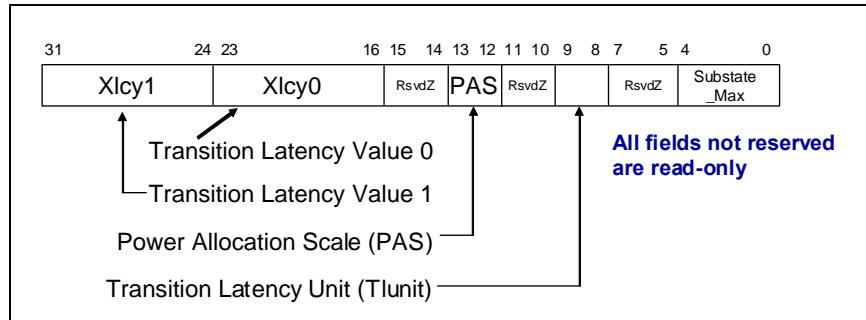
The DPA capability register, shown in Figure 16-4 on page 716, contains several interesting values associated with the substates. The Substate\_Max number indicates how many substates are described, and the numbers must increment contiguously from zero to that value. Two Transition Latency Values are given and each substate will be associated with one or the other by the Latency Indicator register, which contains one bit for each possible substate; if that bit is set Transition Latency Value 1 is used, otherwise Value 0 is used. The latency value gives the maximum time required to transition into that substate from any other

# PCI Express Technology

---

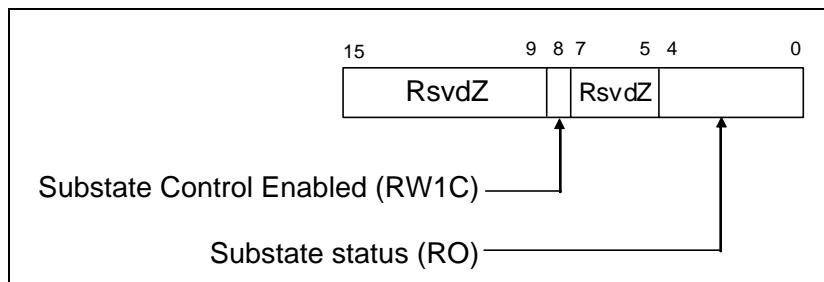
substate. The latencies are multiplied by the Transition Latency Units to give the time in milliseconds. Similarly, the Power Allocation Scale value gives the multiplier for the power used in each substate, expressed in watts. For each defined substate, a 32-bit field in the DPA Power Allocation Array describes the power used for that state. The first one of these is located at offset 010h, and the rest are implemented in subsequent dwords.

Figure 16-4: DPA Capability Register



The low-order five bits of the DPA Control register are written by software to set a new substate, and the current substate can be read from the Status register, as shown in Figure 16-5 on page 716. Notice that bit 8 of the Status register indicates whether the use of DPA substates has been enabled but it's labeled as RW1C (Read, Write 1 to Clear), meaning software can clear this bit but can't set it. DPA is enabled by default after a reset, and software would need to disable it by writing a one to this bit if it did not intend to use DPA.

Figure 16-5: DPA Status Register



## D1 State—Light Sleep

**Optional.** Before going into this state, software must ensure that all outstanding non-posted Requests have received their associated Completions. This can be achieved by polling the Transactions Pending bit in the Device Status register of

# Chapter 16: Power Management

---

the PCI Express Capability block; when the bit is cleared to zero, it's safe to proceed. In this light power conservation state the Function won't initiate Requests except PME Messages, if enabled. Other characteristics of the D1 state include:

- Link is forced to the L1 power state when the Device goes into the D1 state.
- Configuration and Message Requests are accepted in this state, but all other Requests must be handled as Unsupported Requests and all completions may optionally be handled as Unexpected Completions.
- If an error is caused by an incoming Request and reporting it is enabled, an Error Message may be sent while in this state. If a different type of error occurs (such as a Completion timeout), the message won't be sent until the Device is returned to the D0 state.
- The Function may reactivate the Link and send a PME message, if supported and enabled in this state, to notify software that the Function has experienced an event requiring that power be restored.
- The Function may or may not lose its context in this state. If it does and the device supports PME, it must at least maintain its PME context (see "PME Context" on page 710) while in this state.
- The Function must be returned to the D0 Active PM state in order to be fully operational.

Table 16-6 lists the PM policies while in the D1 state.

*Table 16-6: D1 Power Management Policies*

Link PM State	Function PM State	Registers or State that must be valid	Power	Actions permitted to Function	Actions permitted by Function
L1	D1	Device class-specific registers and PME context.*	≤ D0 uninitialized	Config Requests and Messages. Link transitions back to L0 to service the request.	PME Messages.** Though not typically permitted, they would require the Link to transition back to L0.
L2-L3				NA *	

\* This combination of Bus/Function PM states not allowed.

\*\* If PME supported in this state.

## D2 State—Deep Sleep

**Optional.** Before going into this state, software must ensure that all outstanding non-posted Requests have received their associated Completions. This can be achieved by polling the Transactions Pending bit in the Device Status register of

## PCI Express Technology

---

the PCI Express Capability block; when the bit is cleared to zero, it's safe to proceed. This power state provides deeper power conservation than D1 but less than the D<sub>3hot</sub> state. As in D1, the Function won't initiate Requests (except a PME Message) or act as the target of Requests other than configuration. Software must still be able to access the Function's configuration registers in this state.

Other characteristics of the D2 state include:

- Before going into this state, software must ensure that all outstanding non-posted Requests have received their associated Completions. This can be achieved by polling the Transactions Pending bit in the Device Status register of the PCIe Capability block. It could happen that the Completions will never be returned and, in that case, software should wait long enough to ensure they never will be returned.
- Link state must transition to L1 when the Device transitions to the D2 state.
- Configuration and Message Requests are accepted in this state, but all other Requests must be handled as Unsupported Requests and all completions may optionally be handled as Unexpected Completions.
- If an error is caused by an incoming Request and reporting it is enabled, an Error Message may be sent while in this state. If a different type of error occurs (such as a Completion timeout), the message won't be sent until the Device is returned to the D0 state.
- Function may send a PME message, if supported and enabled, to notify software that it needs power restored to handle an event.
- The Function may or may not lose its context in this state. If it does and the device supports PME messages, it must at least maintain its PME context for this purpose.
- The Function must return to the D0 Active state to be fully operational.

Table 16-7 on page 719 illustrates the PM policies while in the D2 state.

# Chapter 16: Power Management

---

Table 16-7: D2 Power Management Policies

Link PM State	Function PM State	Registers and/or State that must be valid	Power	Actions permitted to Function	Actions permitted by Function
L1	D2	Device class-specific registers and PME context.*	≤ next higher supported PM state or ≤ D0 uninitialized.	Config Requests and transactions permitted by device class (typically none). This requires the Link to transition back to L0	PME Messages.* Though not typically permitted, they would require the Link to transition back to L0.
L2/L3		N/A**			

\* If PME supported in this state.

\*\* This combination of Bus/Function PM states not allowed.

## D3—Full Off

**Mandatory.** All Functions must support the D3 state. This is the deepest state and power conservation is maximized. When software writes this power state to the Device, it goes to the **D3<sub>hot</sub>** state, meaning power is still applied. Removing power (Vcc) from the Device puts it into the **D3<sub>cold</sub>** state and the Link into L2, if a secondary power source (Vaux) is available, or L3 if it's not.

**D3<sub>Hot</sub> State. (Mandatory.)** Software puts a Function into D3<sub>hot</sub> by writing the appropriate value into the PowerState field of its Power Mgt Control and Status Register (PMCSR). In this state, the Function can only initiate PME or PME\_TO\_ACK Messages, and can only respond to configuration Requests or the PME\_Turn\_Off Message. Software must be able to access the Function's configuration registers while the device is in the D3<sub>hot</sub> state, if only to be able to change the state back to D0. Other characteristics of D3<sub>hot</sub> include:

- Before going into this state, software must ensure that all outstanding non-posted Requests have received their associated Completions. This can be achieved by polling the Transactions Pending bit in the Device Status register of the PCIe Capability block. It could happen that the Completions will never be returned and, in that case, software should wait long enough to ensure they never will be returned.
- The Link is forced to the L1 state when the Function changes to D3<sub>hot</sub>.

# PCI Express Technology

---

- The Function is allowed to send a PME message to notify PM software of its need to be returned to the fully active state (assuming it supports generation of PM events in the D<sub>3hot</sub> state and has been enabled to do so).
- Function context may be lost when going to this state and if the power is turned off the spec assumes all context will be lost. On the other hand, if the power never goes off before software initiates a return to D0 the context could be maintained. In earlier spec versions that wasn't possible; changing from D<sub>3hot</sub> to D0 involved a soft reset and all the registers were re-initialized. However, the 1.2 revision of that spec added a new capability bit called "No Soft Reset" to indicate that the Function would not do a soft reset in that case. To be able to generate PME messages in the D<sub>3hot</sub> state, a Device must maintain its PME context (see "PME Context" on page 710).

The Function exits from the D<sub>3hot</sub> state under two circumstances:

- If Vcc is removed from the device, it transitions from D<sub>3hot</sub> to D<sub>3cold</sub>.
- Software can write to the PowerState field of the Function's PMCSR register to change its PM state to D0. When programmed to exit D<sub>3hot</sub> and return to D0, the Function returns to the D0 Uninitialized PM state. A reset may or may not be required. Table 16-8 on page 721 lists the PM policies while in the D<sub>3hot</sub> state.

# Chapter 16: Power Management

---

*Table 16-8: D3<sub>hot</sub> Power Management Policies*

Bus PM State	Function PM State	Registers and/or State that must be valid	Power	Actions permitted to Function	Actions permitted by Function
L1	D3 <sub>hot</sub>	PME context. **	≤ next higher supported PM state or ≤ D0 uninitialized.	PCI Express config transactions & PME_Turn_Off broadcast message*** (These can only occur after the Link transitions back to its L0 state.)	PME message** PME_TO_ACK message*** PM_Enter_L23 DLLP***  (These can occur only after the Link returns to L0)
L2/L3 Ready		L2/L3 Ready entered following the PME_Turn_Off handshake sequence, which prepares a device for power removal***			
L2/L3		NA *			

\* This combination of Bus/Function PM states not allowed.

\*\* If PME supported in this state.

\*\*\* See “L2/L3 Ready Handshake Sequence” on page 764 for details regarding the sequence.

**D3<sub>Cold</sub> State. Mandatory.** Every PCI Express Function enters the D3<sub>Cold</sub> PM state upon removal of power (Vcc) from the Function. When power is restored, the device must be reset or generate an internal reset, taking it from D3<sub>Cold</sub> to D0 Uninitialized. A Function capable of generating a PME must maintain PME context while in this state and when transitioning to the D0 state. Since power was removed to arrive at this state, the Function must have an auxiliary power source available if it is to maintain the PME context. Then, when the device goes to D0 Uninitialized, it can generate a PME message to inform the system of a wakeup event, if it’s capable and enabled to do so. For more on auxiliary power, refer to “Auxiliary Power” on page 775.

Table 16-9 on page 722 illustrates the PM policies while in the D3<sub>Cold</sub> state.

# PCI Express Technology

---

Table 16-9:  $D3_{cold}$  Power Management Policies

Bus PM State	Function PM State	Registers and/or State that must be valid	Power	Actions permitted to Function	Actions permitted by Function
L2	$D3_{cold}$	PME context*	AUX Power	Bus reset only	Signal Beacon or WAKE#**
L3		None			None

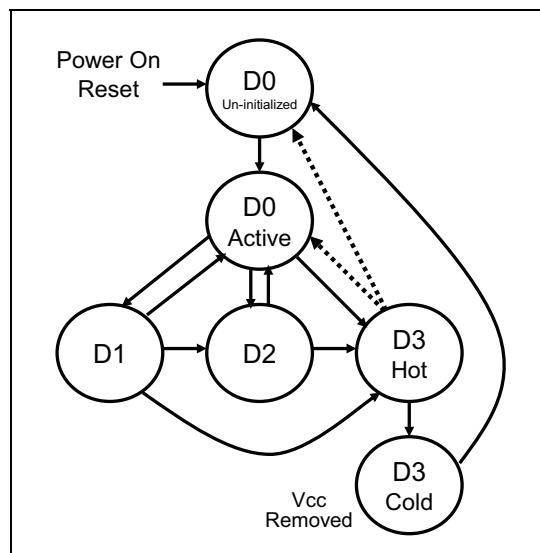
\* If PME supported in this state.

\*\* The method used to signal a wake to restore clock and power depends on the form factor.

## Function PM State Transitions

Figure 16-6 illustrates the PM state transitions for a PCIe Function. Table 16-10 on page 723 provides a description of each transition. Table 16-11 on page 724 illustrates the transitions from one state to another from both a hardware and a software perspective.

Figure 16-6: PCIe Function D-State Transitions



## Chapter 16: Power Management

---

Table 16-10: Description of Function State Transitions

From State	To State	Description
D0 Uninitialized	D0 Active	Function has been completely configured and enabled by its driver.
D0 Active	D1	Software writes the PMCSR PowerState to D1.
	D2	Software writes the PMCSR PowerState to D2.
	D3 <sub>hot</sub>	Software writes the PMCSR PowerState to D3 <sub>hot</sub> .
D1	D0 Active	Software writes the PMCSR PowerState to D0.
	D2	Software writes the PMCSR PowerState to D2.
	D3 <sub>hot</sub>	Software writes the PMCSR PowerState to D3 <sub>hot</sub> .
D2	D0 Active	Software writes the PMCSR PowerState to D0.
	D3 <sub>hot</sub>	Software writes the PMCSR PowerState to D3 <sub>hot</sub> .
D3 <sub>hot</sub>	D3 <sub>cold</sub>	Power is removed from the Function.
	D0 Uninitialized	Software writes the PMCSR PowerState to D0.
D3 <sub>cold</sub>	D0 Uninitialized	Power is restored to the Function.

# PCI Express Technology

---

Table 16-11: Function State Transition Delays

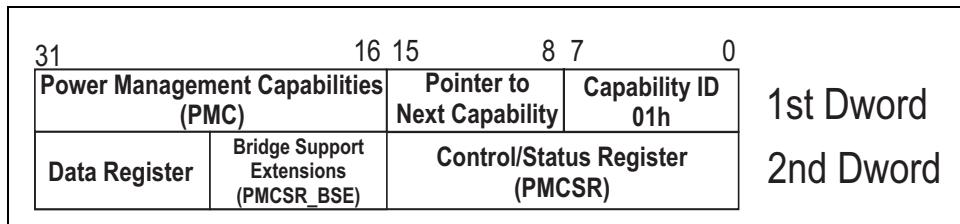
Initial State	Next State	Minimum software-guaranteed delays
D0	D1	0
D0 or D1	D2	200µs from new state setting to first access (including config accesses).
D0, D1, or D2	D3 <sub>hot</sub>	10ms from new state setting to first access.
D1	D0	0
D2	D0	200µs from new state setting to first access.
D3 <sub>hot</sub>	D0	10ms from new state setting to first access.
D3 <sub>cold</sub>	D0	

---

## Detailed Description of PCI-PM Registers

The *PCI Bus PM Interface spec* defines the PM registers (see Figure 16-7) that are implemented in PCIe Functions. Configuration software can determine the PM capabilities and control its properties.

Figure 16-7: PCI Function's PM Registers



### PM Capabilities (PMC) Register

The fields of this 16-bit read-only register are described in Table 16-12.

## Chapter 16: Power Management

*Table 16-12: The PMC Register Bit Assignments*

Bit(s)	Description										
31:27	<p><b>PME_Support</b> field. Indicates in which PM states the Function is capable of sending a PME message. A zero in a bit indicates PME notification is not supported in the respective PM state.</p> <p><u>Bit Corresponds to PM State</u></p> <table><tr><td>27</td><td>D0</td></tr><tr><td>28</td><td>D1</td></tr><tr><td>29</td><td>D2</td></tr><tr><td>30</td><td>D3<sub>hot</sub></td></tr><tr><td>31</td><td>D3<sub>cold</sub> (Function requires aux power for PME logic and Wake signaling via beacon or WAKE# pin) Systems that support wake from D3<sub>cold</sub> must also support aux power and must use it to signal the wakeup. Bits 31, 30, and 27 must be set to 1b for virtual PCI-PCI Bridges imple- mented within Root and Switch Ports. This is required for ports that for- ward PME Messages.</td></tr></table>	27	D0	28	D1	29	D2	30	D3 <sub>hot</sub>	31	D3 <sub>cold</sub> (Function requires aux power for PME logic and Wake signaling via beacon or WAKE# pin) Systems that support wake from D3 <sub>cold</sub> must also support aux power and must use it to signal the wakeup. Bits 31, 30, and 27 must be set to 1b for virtual PCI-PCI Bridges imple- mented within Root and Switch Ports. This is required for ports that for- ward PME Messages.
27	D0										
28	D1										
29	D2										
30	D3 <sub>hot</sub>										
31	D3 <sub>cold</sub> (Function requires aux power for PME logic and Wake signaling via beacon or WAKE# pin) Systems that support wake from D3 <sub>cold</sub> must also support aux power and must use it to signal the wakeup. Bits 31, 30, and 27 must be set to 1b for virtual PCI-PCI Bridges imple- mented within Root and Switch Ports. This is required for ports that for- ward PME Messages.										
26	<b>D2_Support</b> bit. 1 = Function supports the D2 PM state.										
25	<b>D1_Support</b> bit. 1 = Function supports the D1 PM state.										

# PCI Express Technology

---

Table 16-12: The PMC Register Bit Assignments (Continued)

Bit(s)	Description																											
24:22	<p><b>Aux_Current</b> field. For a Function that supports generation of the PME message from the D3<sub>cold</sub> state, this field reports the current demand made upon the 3.3Vaux power source (see “Auxiliary Power” on page 775) by the Function’s logic that retains the PME context information. This information is used by software to determine how many Functions can simultaneously be enabled for PME generation (based on the total amount of current each draws from the system 3.3Vaux power source and the power sourcing capability of the power source).</p> <ul style="list-style-type: none"><li>• If the Function does not support PME notification from within the D3<sub>cold</sub> PM state, this field is not implemented and always returns zero when read. Alternatively, a new feature defined by PCI Express permits devices that do not support PMEs to report the amount of Aux current they draw when enabled by the <i>Aux Power PM Enable</i> bit within the Device Control register.</li><li>• If the Function implements the Data register (see “Data Register” on page 731), this field always returns zeros when read. The Data register then takes precedence over this field in reporting the 3.3Vaux current requirements for the Function.</li><li>• If the Function supports PME notification from the D3<sub>cold</sub> state and does not implement the Data register, then the Aux_Current field reports the 3.3Vaux current requirements for the Function. It is encoded as follows:</li></ul> <table><thead><tr><th>Bit</th><th>24 23 22</th><th>Max Current Required</th></tr></thead><tbody><tr><td></td><td>1 1 1</td><td>375mA</td></tr><tr><td></td><td>1 1 0</td><td>320mA</td></tr><tr><td></td><td>1 0 1</td><td>270mA</td></tr><tr><td></td><td>1 0 0</td><td>220mA</td></tr><tr><td></td><td>0 1 1</td><td>160mA</td></tr><tr><td></td><td>0 1 0</td><td>100mA</td></tr><tr><td></td><td>0 0 1</td><td>55mA</td></tr><tr><td></td><td>0 0 0</td><td>0mA</td></tr></tbody></table>	Bit	24 23 22	Max Current Required		1 1 1	375mA		1 1 0	320mA		1 0 1	270mA		1 0 0	220mA		0 1 1	160mA		0 1 0	100mA		0 0 1	55mA		0 0 0	0mA
Bit	24 23 22	Max Current Required																										
	1 1 1	375mA																										
	1 1 0	320mA																										
	1 0 1	270mA																										
	1 0 0	220mA																										
	0 1 1	160mA																										
	0 1 0	100mA																										
	0 0 1	55mA																										
	0 0 0	0mA																										

# Chapter 16: Power Management

---

Table 16-12: The PMC Register Bit Assignments (Continued)

Bit(s)	Description								
21	<b>Device-Specific Initialization (DSI)</b> bit. A one in this bit indicates that immediately after entry into the D0 Uninitialized state, the Function requires additional configuration above and beyond setup of its PCI configuration Header registers before the Class driver can use the Function. Microsoft OSs do not use this bit. Rather, the determination and initialization is made by the Class driver.								
20	Reserved.								
19	<b>PME Clock</b> bit. Does not apply to PCI Express. Must be hardwired to 0.								
18:16	<b>Version</b> field. This field indicates the version of the PCI Bus PM Interface spec that the Function complies with.  <table><thead><tr><th>Bit</th><th>Complies with Spec Version</th></tr></thead><tbody><tr><td>18 17 16</td><td></td></tr><tr><td>0 0 1</td><td>1.0</td></tr><tr><td>0 1 0</td><td>1.1 (required by PCI Express)</td></tr></tbody></table>	Bit	Complies with Spec Version	18 17 16		0 0 1	1.0	0 1 0	1.1 (required by PCI Express)
Bit	Complies with Spec Version								
18 17 16									
0 0 1	1.0								
0 1 0	1.1 (required by PCI Express)								

## PM Control and Status Register (PMCSR)

This register, required for all PCI Express Devices, serves several purposes as described below. Table 16-13 on page 728 provides a description of the PMCSR bit fields.

- If the Function implements PME capability, a PME Enable bit permits software to enable or disable the Function's ability to assert the PME message or WAKE# signal, and a Status bit reflects whether or not a PME has occurred.
- If the optional Data register is implemented (see "Data Register" on page 731), two fields are used to permit software to select which information can be read through the Data register, and provide the scaling multiplier for the Data register value.
- The register's PowerState field can be read to determine the current PM state of the Function and written to place the Function into a new PM state.

# PCI Express Technology

---

Table 16-13: PM Control/Status Register (PMCSR) Bit Assignments

Bit(s)	Value at Reset	Read/Write	Description
31:24	all zeros	Read Only	See “Data Register” on page 731.
23	zero	Read Only	Not used in PCI Express
22	zero	Read Only	Not used in PCI Express
21:16	all zeros	Read Only	Reserved
15	See Description.	Read, Write one to clear, Sticky RW1CS	<p><b>PME_Status</b> bit. <b>Optional:</b> only implemented if the Function supports PME notification, otherwise zero.</p> <p>This bit reflects whether the Function has experienced a PME (even if the PME_En bit in this register has disabled the Function’s ability to send a PME message). If set to one, the Function has experienced a PME. Software clears this bit by writing a one to it.</p> <p>After reset, this bit is zero if the Function doesn’t support PME in D3<sub>cold</sub>. If the Function does support PME in D3<sub>cold</sub>, this bit is indeterminate at initial OS boot time but after that reflects whether the Function has experienced a PME.</p> <p>If the Function supports PME from D3<sub>cold</sub>, the state of this bit must persist even if power is lost or the Function is reset (a sticky bit). This implies that an auxiliary power source keeps this logic active during these conditions (see “Auxiliary Power” on page 775).</p>

## Chapter 16: Power Management

---

Table 16-13: PM Control/Status Register (PMCSR) Bit Assignments (Continued)

Bit(s)	Value at Reset	Read/Write	Description
14:13	Device-specific	Read Only	<p><b>Data_Scale</b> field. <b>Optional</b>. If the Function does not implement the Data register this field is hardwired to return zeros.</p> <p>If the Data register is implemented, the Data_Scale field is mandatory and must be a read-only value representing the multiplier for it. The value and interpretation of the Data_Scale field depends on the data item selected to be viewed through the Data register by the Data_Select field.</p>
12:9	0000b	Read/Write	<p><b>Data_Select</b> field. <b>Optional</b>. If the Function does not implement the Data register, this field is hardwired to return zeros.</p> <p>If the Data register is implemented, Data_Select is a mandatory read/write field. The value placed in this register selects the data to be viewed in the Data register. That value must then be multiplied by the value read from the Data_Scale field.</p>

# PCI Express Technology

---

Table 16-13: PM Control/Status Register (PMCSR) Bit Assignments (Continued)

Bit(s)	Value at Reset	Read/Write	Description										
8	See Description.	Read/Write	<p><b>PME_En</b> bit. <b>Optional</b>.          1 = enable Function's ability to send PME messages when an event occurs.          0 = disable.</p> <p>If the Function does not support the generation of PMEs from any power state, this bit always return zero when read.</p> <p>After reset, this bit is zero if the Function doesn't support PME from D3<sub>cold</sub>. If the Function supports PME from D3<sub>cold</sub>:</p> <ul style="list-style-type: none"> <li>• this bit is indeterminate at initial OS boot time.</li> <li>• otherwise, it enables or disables whether the Function can send a PME message in case a PME occurs.</li> </ul> <p>If the Function supports PME from D3<sub>cold</sub>, the state of this bit must persist while the Function remains in the D3<sub>cold</sub> state and during the transition from D3<sub>cold</sub> to the D0 Uninitialized state. This implies that the PME logic must use an aux power source to power this logic during these conditions.</p>										
7:2	all zeros	Read Only	Reserved										
1:0	00b	Read/Write	<p><b>PowerState</b> field. <b>Mandatory</b>. Software uses this field to read the current PM state of the Function or write a new PM state. If software selects a PM state not supported by the Function, the write completes normally but the data is discarded and no state change occurs.</p> <table style="margin-left: 20px;"> <tr> <th colspan="2"><u>1 0      PM State</u></th> </tr> <tr> <td>0 0</td> <td>D0</td> </tr> <tr> <td>0 1</td> <td>D1</td> </tr> <tr> <td>1 0</td> <td>D2</td> </tr> <tr> <td>1 1</td> <td>D3<sub>hot</sub></td> </tr> </table>	<u>1 0      PM State</u>		0 0	D0	0 1	D1	1 0	D2	1 1	D3 <sub>hot</sub>
<u>1 0      PM State</u>													
0 0	D0												
0 1	D1												
1 0	D2												
1 1	D3 <sub>hot</sub>												

# **Chapter 16: Power Management**

---

## **Data Register**

**Optional, read-only.** Refer to Figure 16-8 on page 732. The Data register is an 8-bit, read-only register that provides software with the following information:

- Power consumed in the selected PM state; useful in power budgeting.
- Power dissipated in the selected PM state; useful in managing the thermal environment.
- Any type of data could be reported through this register, but the PCI-PM spec only defines power consumption and power dissipation information for it.

If the Data register is implemented, the Data\_Select and Data\_Scale fields of the PMCSR registers must also be implemented, and the Aux\_Current field of the PMC register must not be implemented.

**Determining Presence of the Data Register.** Software can perform the following procedure to check for the presence of the Data register:

1. Write a value of 0000b into the Data\_Select field of the PMCSR register.
2. Read from either the Data register or the Data\_Scale field of the PMCSR register. A non-zero value indicates that the Data register as well as the Data\_Scale and Data\_Select fields of the PMCSR registers are implemented. If a value of zero is read, go to step 4.
3. If the current value of the Data\_Select field is a value other than 1111b, go to step 4. If the current value of the Data\_Select field is 1111b, all possible Data register values have been scanned and returned zero, indicating that neither the Data register nor the Data\_Scale and Data\_Select fields of the PMCSR registers are implemented.
4. Increment the content of the Data\_Select field and go back to step 2. Since the data select field is only 4 bits, a complete scan requires testing 16 possible select values and looking to see if any non-zero values are seen for the data and scale registers.

**Operation of the Data Register.** The information returned is typically a static copy of the Function's worst-case power consumption and power dissipation characteristics in the various PM states (as listed in the Device's data sheet). To use the Data register, the programmer uses the following sequence:

1. Write a value into the Data\_Select field (see Table 16-14 on page 733) of the PMCSR register to select the data item to be viewed through the Data register.

# PCI Express Technology

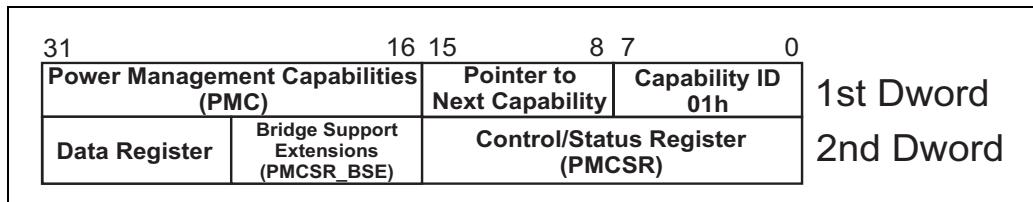
---

2. Read the data value from Data register and the Data\_Scale field of the PMCSR register.
3. Multiply the value by the scaling factor.

**Multi-Function Devices.** In a multi-function PCI Express device, each Function must supply its own power information. The power information for the logic common to all the Functions is reported through Function zero's Data register (see Data Select Value = 8 in Table 16-14 on page 733).

**Virtual PCI-to-PCI Bridge Power Data.** The spec doesn't specify data field use in PCI-to-PCI bridge Functions in a Root Complex or Switch. But, to maintain PCI-PM compatibility, bridges must report the power information they consume. Software could read the virtual PPB Data registers at each port of a switch to determine the power consumed by the switch in each power state.

Figure 16-8: PM Registers



# Chapter 16: Power Management

---

Table 16-14: Data Register Interpretation

Data Select Value	Data Reported in Data Register	Interpretation of Data Scale Field in PMCSR	Units/Accuracy
00h	Power consumed in D0		
01h	Power consumed in D1		
02h	Power consumed in D2		
03h	Power consumed in D3		
04h	Power dissipated in D0		
05h	Power dissipated in D1		
06h	Power dissipated in D2		
07h	Power dissipated in D3		
08h	In a multi-function PCI device, Function 0 indicates power consumed by logic common to all Functions in the package.	00b = unknown 01b = multiply by 0.1 10b = multiply by 0.01 11b = multiply by 0.001	Watts
09h-0Fh	Reserved for future use of Function 0 in a multi-function device.		
08h-0Fh	Reserved in single-function devices and Functions other than Function 0 in a multi-function device	Reserved	TBD

---

## Introduction to Link Power Management

We've just seen how software can put Devices into one of several device power states, now let's consider how PCIe also manages Link power. Device power and Link power are related to each other, as shown in Table 16-15 on page 734. Note also the relationship between downstream and upstream devices, which can be summarized by saying that an upstream Device or Link cannot be in a more aggressive power-conserving state than the one below it. The reason is to

# PCI Express Technology

---

facilitate timely delivery of packets from the Endpoints, whose traffic would be delayed if upstream devices were in a lower power state. Each relationship is described below:

**D0** — Device is fully powered and typically in the L0 Link state. Some power conservation is available without leaving this state by using DPA substates (see “Dynamic Power Allocation (DPA)” on page 714), and by using the hardware-based Link power management (see “Active State Power Management (ASPM)” on page 735 for more details).

**D1 & D2** — When software changes the device state to D1 or D2, the Link must automatically transition to the L1 state. Since both Link partners are involved in this operation there is a handshake mechanism to ensure that things are done in an orderly fashion.

**D3<sub>hot</sub>** — When software places a device into the D3 state, the Link automatically transitions to L1 just as it does when going to the D1 and D2 states. Software may now choose to remove the reference clock and power, putting the device into D3<sub>cold</sub>. But, before doing that, it's expected that the system will initiate a handshake process to prepare the Links by putting them into the L2/L3 Ready state.

**D3<sub>cold</sub>** — In this state, main power and the reference clock have been turned off. However, auxiliary power ( $V_{AUX}$ ) may be available, allowing the device to signal a wakeup event to the system. If it is, the Link state will be in L2. If main power is removed but  $V_{AUX}$  is not available, the Link will be in L3. Table 16-16 on page 735 provides additional information regarding the Link power states.

*Table 16-15: Relationship Between Device and Link Power States*

Downstream Component D-State	Permissible Upstream Component D-State	Permissible Interconnect State
D0	D0	L0, L0s & L1 (optional)
D1	D0-D1	L1
D2	D0-D2	L1
D3 hot	D0-D3 hot	L1, L2/L3 Ready
D3 cold	D0-D3 cold	L2 (AUX Pwr), L3

# Chapter 16: Power Management

---

Table 16-16: Link Power State Characteristics

State	Description	Software Directed?	Active State Link PM	Ref. Clocks	Main Power	PLL	Vaux
L0	Fully Active	Yes (D0)	On	On	On	On	On/Off
L0s	Standby	No	Yes (D0)	On	On	On	On/Off
L1	Low Power Standby	Yes* (D1-D3 hot)	Yes (option) (D0)	On	On	On/Off	On/Off
L2/L3 Ready	Staging for power removal	Yes PME_Turn_Off handshake	No	On	On	On/Off	On/Off
L2	Low Power Sleep	Yes**	No	Off	Off	Off	On
L3	Off (Zero Power)	N/A	N/A	Off	Off	Off	Off

\* The L1 state is entered either due to PM software placing a device into the D1, D2, or D3 states or under hardware control with ASPM.

\*\* The spec describes the L2 state as being software directed. The other L-states in the table are listed as software directed because software initiates the transition into these states. For example, when software initiating a device power state change to D1, D2, or D3 devices must respond by entering the L1 state. Software then causes the transition to the L2/L3 Ready state by initiating a PME\_Turn\_Off message. Finally, software initiates the removal of power from a device after the device has transitioned to the L2/L3 Ready state. Because Vaux power is available in L2, a wakeup event can be signaled to notify software.

---

## Active State Power Management (ASPM)

ASPM is a hardware-based Link power conservation mechanism that only applies while the device is in the D0 device power state. Transitions into and out of ASPM states are initiated by hardware based on implementation-specific criteria; software can't control or observe this operation, it can only enable or disable it using configuration register bits (see Figure 16-15 on page 744).

Two low power states are defined for ASPM:

1. L0s (standby state) — This state provide substantial power savings but still allows quick entry and exit latencies. The main way this is done is by putting the Transmitter into the Electrical Idle condition. Support for this state was previously required for all PCIe devices in the earlier spec versions, but in the 3.0 spec it became optional.
2. L1 ASPM — The goal for L1 is to achieve greater power conservation than L0s for situations where longer entry and exit latencies are acceptable. For example, in this state both Transmitters go into Electrical Idle at the same time. Support for this state continues to be optional in the 3.0 spec as it was in the earlier specs.

---

## Electrical Idle

Since putting a Transmitter into Electrical Idle is a central part of ASPM, it will help to discuss how doing so works. When a Transmitter's differential signals (Tx<sub>D+</sub> and Tx<sub>D-</sub>) goes into the Electrical Idle condition, it stops signaling and instead holds its voltage very close to the common mode voltage with a differential voltage of 0 V. Signal transitions consume power, so stopping them on the Link gives power savings while still allowing a fairly quick resumption back to normal Link activity during which it is said to be in the L0 state. Depending on the degree of power savings, the Link is either in the L0s or L1 state. During this time, the transmitter may choose to remain in the low-impedance state or change to high impedance by turning off its termination logic to save more power. In addition to L0s and L1, Electrical Idle will also be in effect when the Link has been disabled.

### Transmitter Entry to Electrical Idle

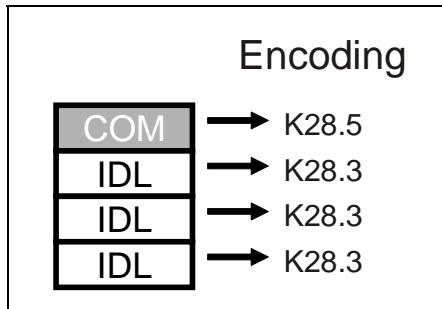
Transmitters that wish to enter the Electrical Idle condition must first inform the Link partner so the lack of further signaling won't be misinterpreted as an error. They do that by sending the EIOS (Electrical Idle Ordered-Set) and then quickly ceasing transmission and tri-stating the Link output drivers. What the EIOS looks like depends on the encoding method in use, as described in the following sections. Once the last EIOS has been sent, the Transmitter must enter Electrical Idle within 8ns and remain in that mode for at least 20ns, regardless of the data rate. The differential peak voltage allowed during Electrical Idle must be between 0 and 20mV peak, again regardless of the data rate, to reduce the chance of the Receiver misinterpreting noise on the line as a valid signal. (See Table 13-3 on page 489 for more on these timing and voltage parameters.)

# Chapter 16: Power Management

---

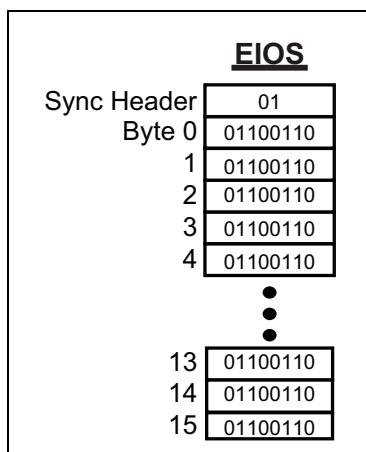
**Gen1/Gen2 Mode Encoding.** For Gen1/Gen2 mode, the EIOS takes the form shown in Figure 16-9 on page 737. All four Symbols must be sent, but the Receiver only needs to see two IDL control characters to recognize this condition.

Figure 16-9: Gen1/Gen2 Mode EIOS Pattern



**Gen3 Mode Encoding.** For Gen3 mode, the EIOS is an Ordered Set block that consists of an Ordered Set Sync Header (01b) followed by 16 bytes that are all 66h, as shown in Figure 16-10 on page 737. Curiously, a Transmitter is not required to finish the block if it will go directly to Electrical Idle but is allowed to stop after Symbol 13 (anywhere in Symbol 14 or 15). The reason is to allow for the case where an internal clock doesn't line up with the Symbol boundaries due to 128b/130b encoding. This truncation won't cause a problem at the Receiver because it only needs to see Symbols 0 - 3 of the EIOS to recognize it.

Figure 16-10: Gen3 Mode EIOS Pattern



## Transmitter Exit from Electrical Idle

When a Transmitter is instructed to exit from Electrical Idle, the steps it takes depend on the data rate in use (see below). However, it must resume transmission within less than 8ns by sending FTSs or TS1/TS2s causing transition back to the L0 full-on state.

**Gen1 Mode.** For 2.5 GT/s, the process is simple: it begins using valid differential signals to send the TS1s or FTSs that will serve to inform the Receiver about the change. The Receiver detects the voltage as being above the squelch threshold and begins to evaluate the incoming signal.

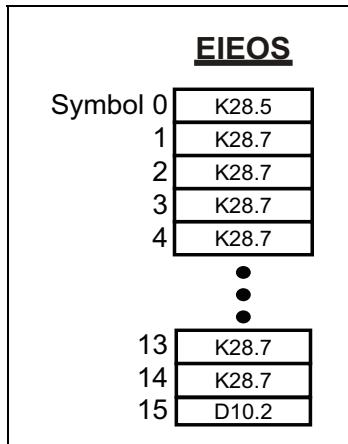
**Gen2 Mode.** When using 5.0 GT/s, the signals are changing so quickly that they don't have time to reach the higher voltage levels. That makes it more difficult to quickly detect when the voltages have changed back to the operational values. To make this easier, the EIEOS (Electrical Idle Exit Ordered Set), was defined to provide a lower-frequency sequence. The EIEOS for 8b/10b encoding, shown in Figure 16-11 on page 739, uses repeated K28.7 control characters to appear as a repeating string of 5 ones followed by 5 zeros. This gives the low-frequency signal that allows the higher signal voltages that are more readily seen. In fact, the spec states that this pattern guarantees that the Receiver will properly detect an exit from Electrical Idle, something that scrambled data cannot do. The EIEOS is to be sent under the following conditions:

- Before the first TS1 after entering the Configuration.Linkwidth.Start or Recovery.RcvrLock state.
- After every 32 TS1s or TS2s are sent in Configuration.Linkwidth.Start, Recovery.RcvrLock, or Recovery.RcvrCfg states. The TS1/TS2 count is reset to zero whenever an EIEOS is sent or the first TS2 is received in the Recovery.RcvrCfg state.

# Chapter 16: Power Management

---

Figure 16-11: Gen1/Gen2 Mode EIEOS Symbol Pattern



**Gen3 Mode.** An EIEOS is needed for 8 GT/s rate too and for the same reason as for 5.0 GT/s. Now, though, the Ordered Set takes the form of a block, as shown in Figure 16-12 on page 740. As before, it gives a low-frequency pattern in alternating bytes of 00h and FFh, which appears as a repeating string of 8 zeros followed by 8 ones.

In addition, EIEOS is sent so as to allow a receiver during LTSSM Recovery state to establish Block Lock after which the Link transitions to the L0 state. See the section “Block Alignment” on page 411 and “Achieving Block Alignment” on page 438.

In Gen3 mode, EIEOS is to be sent:

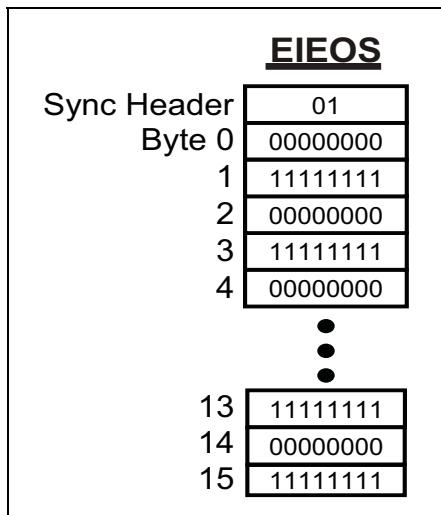
- Before the first TS1 after entering the Configuration.Linkwidth.Start or Recovery.RcvrLock state.
- Immediately after an EDS Framing Token when a Data Stream is ending if an EIOS is not being sent and the LTSSM is not entering Recovery.RcvrLock.
- After every 32 TS1s/TS2s whenever TS1s or TS2s are sent. The count is reset to zero when:
  - an EIEOS is sent
  - the first TS2 is received while in either the Recovery.RcvrCfg or Configuration.Complete LTSSM state
  - a Downstream Port in Phase 2 of the Equalization sequence, or an Upstream Port in Phase 3, receives two TS1s with the Reset EIEOS Interval Count bit set.

# PCI Express Technology

---

- After every  $2^{16}$  TS1s during the Equalization sequence, if the Reset EIEOS Interval Count bit has prevented it from being sent. The spec states that designs are allowed to satisfy this requirement by sending an EIEOS within 2 TS1s of the scrambling LFSR matching its seed value.
- As part of an FTS sequence, Compliance Pattern, or Modified Compliance pattern.

Figure 16-12: 128b/130b EIEOS Block



## Receiver Entry to Electrical Idle

When a Transmitter enters Electrical Idle, the Link partner's Receiver responds based on the data rate, as described in the following sections. Receipt of an EIOS informs the Receiver that this is going to happen, preparing it to detect when it actually does happen. When the Receiver detects this condition it de-gates the error logic to prevent reporting errors caused by unreliable activity on the Link and arms its Electrical Idle Exit detector so it will be ready to resume normal activity when the Transmitter begins to send data again. There are two Electrical Idle detection options.:

**Detecting Electrical Idle Voltage.** Once an EIOS has been received, the expectation is that the Transmitter will cease transmission very quickly. In the 1.x spec versions Receivers detect this by observing that the incoming voltage has dropped below the threshold of a valid signal. This isn't too difficult at 2.5 GT/s but it requires a squelch detect circuit that consumes space and power.

# Chapter 16: Power Management

---

**Inferring Electrical Idle.** However, at higher frequencies the signal becomes increasingly attenuated, making it difficult for squelch detect logic to distinguish the levels. This is especially true for 8.0 GT/s, where it's expected that the Receiver may need to perform equalization internally to recover a good signal. To alleviate these detection problems, the 2.0 spec introduced the concept of allowing a Receiver to infer when the Link has gone to the Electrical Idle condition rather than testing the voltage level. In this model, the absence of expected events is used to indicate that the Link is not signaling and can therefore be assumed to be in Electrical Idle, as listed in Table 16-17. By way of explanation, Flow Control Updates should arrive regularly while the Link is in L0, and SOFs are expected with certain timing, too. For simplicity, a Receiver is allowed to check for one or the other or both of these conditions. During Link training the TS1s and TS2s should arrive regularly, so their absence can also be taken to mean that the Link is Idle. For the last two rows of the table, though, it's possible that no Symbols have been received at all, and that will also be understood to mean the Link is Idle. Since Electrical Idle takes place for the overall Link and not for Lanes independently, there's no need for each Lane to measure these times. Instead, an LTSSM can just use one timer in common for all the Lanes on that Link.

*Table 16-17: Electrical Idle Inference Conditions*

State	2.5GT/s	5.0 GT/s	8.0 GT/s
L0	Absence of an FC Update or SOS in a 128μs window		
Recovery.RcvrCfg	Absence of a TS1 or TS2 in a 1280 UI interval		Absence of a TS1 or TS2 in a 4ms window
Recovery.Speed (successful_speed_negotiation = 1b)	Absence of a TS1 or TS2 in a 1280 UI interval		Absence of a TS1 or TS2 in a 4680 UI interval
Recovery.Speed (successful_speed_negotiation = 0b)	Absence of an exit from Electrical Idle in a 2000 UI interval	Absence of an exit from Electrical Idle in a 16000 UI interval	
Loopback.Active (as a slave)	Absence of an exit from Electrical Idle in a 128μs window	N/A	N/A

# PCI Express Technology

---

How the EIOS is recognized at the Receiver also depends on the encoding scheme. For Gen1/Gen2 mode, a receiver recognizes an EIOS when it sees two of the three IDL Symbols. For Gen3 mode, it's recognized when Symbols 0-3 of the incoming block match the EIOS pattern.

## Receiver Exit from Electrical Idle

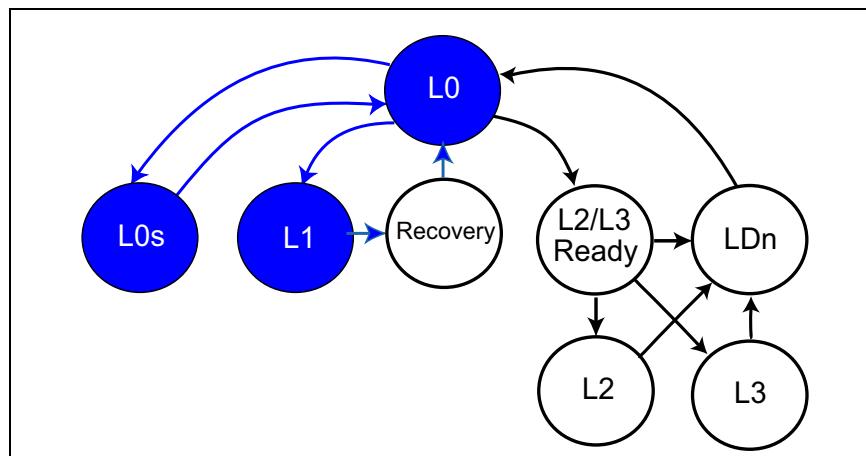
Receivers detect a voltage difference to signify a resumption of normal signaling. An exit from Electrical Idle will be detected when the differential peak-to-peak voltage exceeds the Electrical Idle Detect threshold, which is allowed to be set between 65 and 175mV for all data rates.

At 2.5 GT/s nothing more is needed, but at higher rates Receivers don't have to rely on this detection circuit except when receiving EIEOS during certain LTSSM states or during the four EIE Symbols that precede transmission of an FTS sequence at 5.0 GT/s. The number and timing of EIEOSs to facilitate detection of Electrical Idle exit depends on the Link state. For more on this, see "Active State Power Management (ASPM)" on page 735.

In Electrical Idle, the Receiver's PLL loses clock synchronization. When the Transmitter exits Electrical Idle, it sends FTSs to exit from L0s, or TS1/TS2s to exit from all other Link states. Doing so supplies the needed transition density for the CDR logic to re-synchronize the receiver PLL and achieve Bit Lock and Symbol Lock or Block Alignment.

Figure 16-13 illustrates the Link state transitions and highlights the transitions between L0, L0s, and L1. Note that there is no direct path from L0s to L1, so the Link must be returned to the L0 state before changing between them.

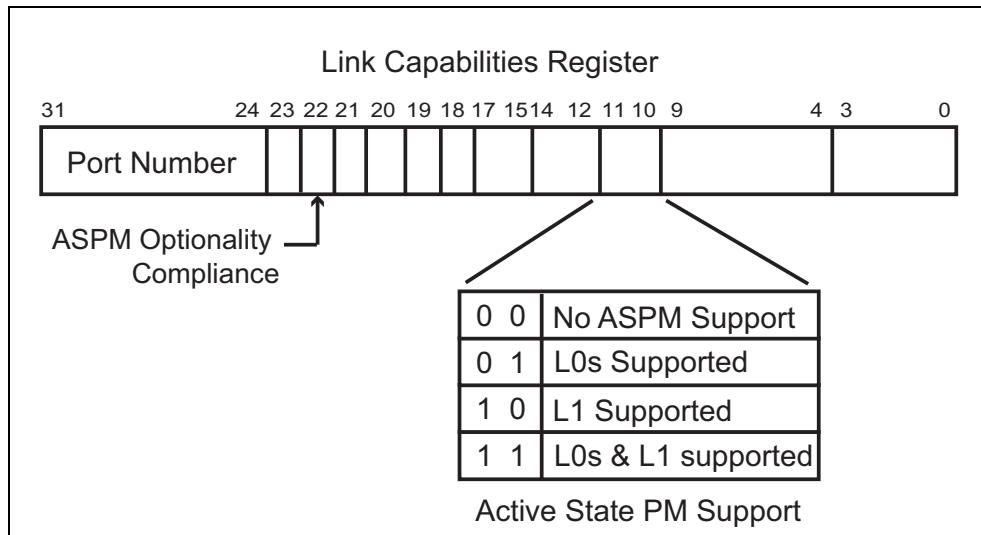
Figure 16-13: ASPM Link State Transitions



# Chapter 16: Power Management

The Link Capability register specifies a device's support for Active State Power Management. Figure 16-14 illustrates the *ASPM Support* field within this register. In earlier spec versions, not all 4 options were available, but the 2.1 spec filled in all of them. Note that bit 22 indicates whether all the options are available.

Figure 16-14: ASPM Support



Software can enable and disable ASPM via the *Active State PM Control* field of the Link Control Register as illustrated in Figure 16-15 on page 744. The possible settings are listed in Table 16-18 on page 743. Note: The spec recommends that ASPM be disabled for all components in a path used for Isochronous transactions if the additional latencies associated with ASPM exceed the limits of the isochronous transactions.

Table 16-18: Active State Power Management Control Field Definition

Setting	Description
00b	L0s and L1 ASPM disabled
01b	L0s enabled and L1 disabled

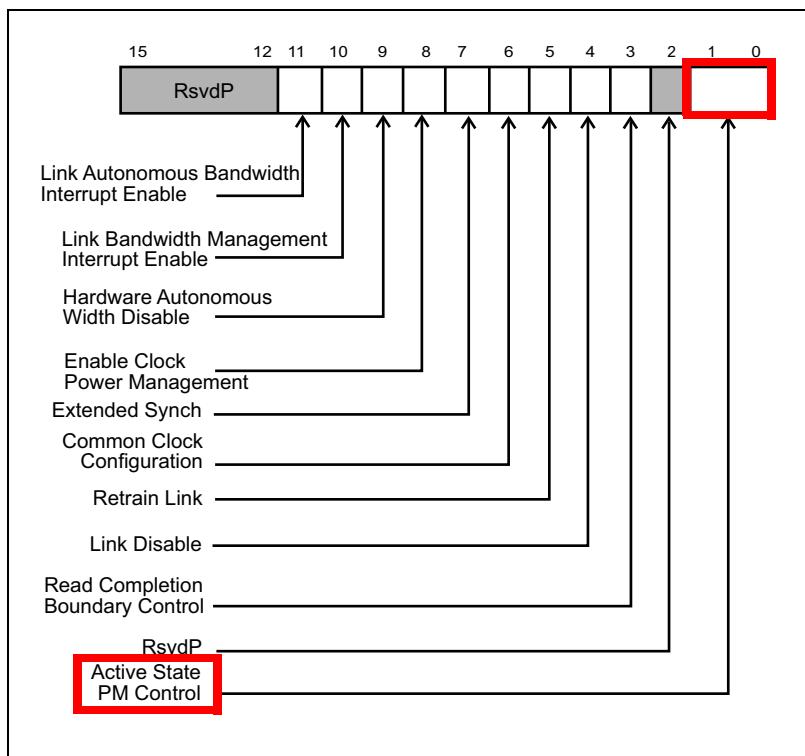
# PCI Express Technology

---

Table 16-18: Active State Power Management Control Field Definition (Continued)

Setting	Description
10b	L1 enabled and L0s disabled
11b	Both L0s and L1 enabled

Figure 16-15: Active State PM Control Field



---

## L0s State

L0s is a Link power state that can only be entered under hardware control and is applied to a single direction of the Link. For example, a large volume of traffic in conventional PC-based systems results from Functions sending data to main system memory. As a result, the upstream lanes carry heavy traffic while the downstream lanes may carry very little. These downstream lanes can enter the L0s state to conserve power during stretches of idle bus time.

# Chapter 16: Power Management

## Entry into L0s

A Transmitter initiates a change from L0 to L0s after detecting a period of idle time that is implementation specific.

**Entry into L0s.** Entry is managed for a single direction of the Link based on detecting a period of Link idle time. Ports are required to enter L0s after detecting idle time of no greater than  $7\mu s$ .

Idle is defined differently for Endpoints and Switches. The reason for this is a desire to minimize recovery time as Link recovery time propagates through Switches. For example, if a Switch upstream port was in a low power state and now sees activity, it means that a TLP is probably on its way down to the Switch. Where will the packet need to be routed? It will go to one of the downstream ports, but rather than wait to receive the packet and determine which port will be the target before starting to wake it up, the lowest-latency approach would be to wake all the downstream ports so that the one that turns out to be the target will be ready as quickly as possible.

Basic rules regarding idle time:

- **Endpoint Port or Root Port:**
  - No TLPs are pending transmission or a lack of Flow Control credits is temporarily blocking them.
  - No DLLPs are pending transmission.
- **Upstream Switch Port:**
  - The receive lane of all downstream ports are already in L0s.
  - No TLPs are pending transmission or a lack of Flow Control credits is temporarily blocking them.
  - No DLLPs are pending transmission.
- **Downstream Switch Port:**
  - The Switch's Upstream Port's Receive Lanes are in L0s.
  - No TLPs are pending transmission or a lack of Flow Control credits is temporarily blocking them.
  - No DLLPs are pending for transmission

The Transaction and Data Link Layers are unaware of whether the Physical Layer transmitter has entered L0s, but the idle conditions that trigger a transition to L0s must be continuously reported from the Transaction and Link layers to the Physical Layer so it can make timely choices about this. Note that a port must always tolerate L0s on its receiver, even if software has disabled ASPM. This allows a device at the other end of the Link that is enabled for ASPM to still transition one side of the Link to the L0s state.

**Flow Control Credits Must be Delivered.** One situation that qualifies as idle time is a pending TLP that is blocked due to insufficient FC credits. When flow control credits are received that allow delivery of the pending TLP, the transmitting port must initiate a return to L0. Also, if the receive buffer associated with the transmitter in L0s makes additional flow control credits available, the transmitter must return to L0 and deliver the FC\_Update DLLP to the neighbor.

**Transmitter Initiates Entry to L0s.** When sufficient idle time has been observed by a Transmitter, it forces a transition from L0 to L0s by sending an “electrical idle” ordered set (EIOS) to the receiver and stopping transmission. The transmitter and receiver are now in their electrical idle states and have reduced power consumption. Synchronization between the transmitter and receiver has been lost and retraining will be required for recovery. The spec requires that the PLL logic in the receiver must remain active (powered) to allow quick recovery from L0s back to L0.

## Exit from L0s State

If the transmitter detects that the idle condition is no longer true, it must initiate the exit from L0s to L0. The spec encourages designers to monitor events that give an early indication that an L0s exit is imminent and start the recovery process to speed up the transition back to L0. For example, if the Receiver of the port receives a non-posted Request, the Transmitter knows that it will soon be asked to send a Completion in response. Consequently, the Transmitter can go ahead and start the exit process so the Link state is L0 by the time it is asked to deliver the Completion.

**Transmitter Initiates L0s Exit.** To exit L0s, the Transmitter sends one or more Fast Training Sequence (FTS) Ordered Sets. The number of these required by the Link partner’s Receiver was communicated earlier during Link training (N\_FTS field in the TS1s and TS2s used in training). After sending the requested number of FTSs, one SOS is delivered. The receiver should be able to establish bit lock and symbol lock or Block lock, and should be ready to resume normal operation.

**Actions Taken by Switches that Receive L0s Exit.** A switch that receives an L0s to L0 transition sequence on one port may also need to initiate an L0s exit to other of its ports. Two specific cases are considered:

- **Switch Downstream Port Receives L0s to L0 transition.** The switch must signal an L0s to L0 on its upstream port if it is currently in the L0s state because the packet coming up from the Endpoint or downstream switch will most likely need to go upstream to the Root Complex.

# Chapter 16: Power Management

---

- **Switch Upstream Port Receives L0s to L0 transition.** The switch must signal an L0s to L0 transition on all downstream ports currently in the L0s state because it doesn't want to wait until the packet arrives to begin waking the target path.

Switch ports that were put into L1 by a software change to the device power state remain unaffected by L0s to L0 transitions. However, once the upstream Link has completed the transition to L0, a subsequent transaction may target this port, causing a transition from L1 to L0.

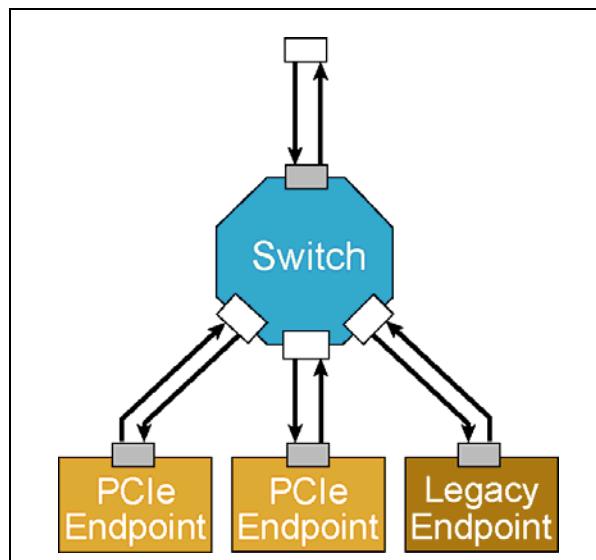
---

## L1 ASPM State

The optional L1 ASPM state provides deeper power savings than L0s, but has a greater recovery latency. This state results in both directions of the Link going into the L1 state and results in Link and Transaction layer deactivation within each device.

Entry into this state is requested by an upstream port, such as from an Endpoint or the upstream port of a switch (upstream ports are shaded as shown in Figure 16-16). The downstream port responds to this request and either agrees to go into L1 or rejects the request through a negotiation process with the downstream component. Exiting L1 ASPM can be initiated by either the downstream or upstream port.

*Figure 16-16: Only Upstream Ports Initiate L1 ASPM*



## Downstream Component Decides to Enter L1 ASPM

The spec does not precisely define all conditions under which an Endpoint or upstream port of a switch decides to attempt entry into the L1 ASPM state but does suggest that one case might be when both sides of the Link have been in L0s for a preset amount of time. The requirements given include:

- ASPM L1 entry is supported and enabled
- Device-specific requirements for entering L1 have been satisfied
- No TLPs are pending transmission
- No DLLPs are pending transmission
- If the downstream component is a switch, then all of the switch's downstream ports must be in the L1 or higher power conservation state before the upstream port can initiate L1 entry.

## Negotiation Required to Enter L1 ASPM

Because of the longer latency required to recover from L1 ASPM, a negotiation process is employed to ensure that the port at the other end of the Link is enabled for L1 ASPM and is prepared to enter it. The negotiation involves sending several packets:

- PM\_Active\_State\_Request\_L1 DLLP — issued by the downstream port to start the negotiation process.
- PM\_Request\_Ack DLLP — returned by the upstream port when all of its requirements to enter L1 ASPM have been satisfied.
- PM\_Active\_State\_Nak message TLP — returned by the upstream port when it is unable to enter the L1 ASPM state.

The upstream component may or may not accept the transition to the L1 ASPM state. The following scenarios describe a variety of circumstances that result in both conditions.

### Scenario 1: Both Ports Ready to Enter L1 ASPM State

Figure 16-17 on page 750 summarizes the sequence of events that must occur to enable transition to the L1 ASPM state. This scenario assumes that all transactions have completed in both directions and no new transaction requirements emerge during the negotiation.

**Downstream Component Requests L1 State.** If the downstream component wishes to transition to the L1 state, it can send the request to enter L1 after the following steps have completed:

# **Chapter 16: Power Management**

---

1. TLP scheduling is blocked at the Transaction Layer.
2. The Link Layer has received acknowledgement for the last TLP it had previously sent and the replay buffer is empty.
3. Sufficient flow control credits are available to allow transmission of the largest possible packet for any FC type. This ensures that the component can issue a TLP immediately upon exiting the L1 state.

The downstream component then delivers the PM\_Active\_State\_Request\_L1 to notify the upstream component of the request to enter the L1 state. This is sent repeatedly until the upstream component responds — either a PM\_Request\_ACK DLLP or a PM\_Active\_State\_NAK message.

**Upstream Component Response to L1 ASPM Request.** Downstream ports (i.e., ports of an upstream component that face downward) must accept a request to enter a low power L1 state if all of the following conditions are true:

- The Port supports ASPM L1 entry and is enabled to do so
- No TLP is scheduled for transmission
- No Ack or Nak DLLP is scheduled for transmission

**Upstream Component Acknowledges Request to Enter L1.** The upstream component sends a PM\_Request\_ACK to notify the downstream component of its agreement to enter the L1 ASPM state after it:

1. Block scheduling of any new TLPs.
2. Receive acknowledgement for the last TLP previously sent (meaning its replay buffer is empty).
3. Ensure enough flow control credits are available to send the largest possible packet for any FC type so that it can issue a TLP immediately after exiting the L1 state.

The Upstream component then sends PM\_Request\_Ack continuously until it detects the EIOS on its receive lanes, indicating that the downstream device has entered Electrical Idle.

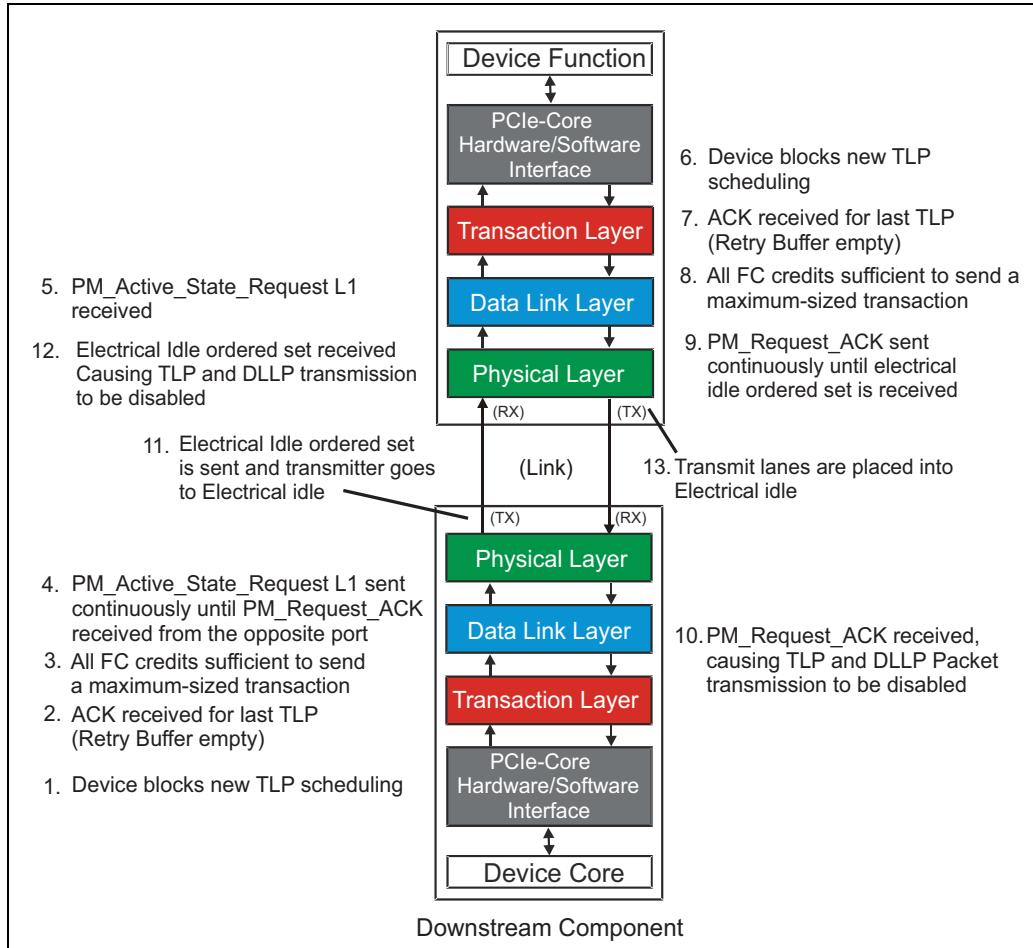
**Downstream Component Sees Acknowledgement.** When the Downstream component sees the PM\_Request\_Ack, it stops sending the PM\_Active\_State\_Request\_L1, disables DLLP and TLP transmission, sends the EIOS and places its transmit lanes into Electrical Idle.

**Upstream Component Receives Electrical Idle.** When the Upstream component receives the EIOS, it stops sending the PM\_Request\_Ack DLLP,

# PCI Express Technology

disables DLLP and TLP transmission, sends EIOS and places its own transmit lanes into Electrical Idle.

Figure 16-17: Negotiation Sequence Required to Enter L1 Active State PM



## Scenario 2: Upstream Component Transmits TLP Just Prior to Receiving L1 Request

This scenario presumes that the upstream component has just been instructed by its core logic to send a TLP downstream before it receives the request to enter L1 from the downstream device. Several negotiation rules define the actions to ensure that this situation is managed correctly.

# Chapter 16: Power Management

---

**TLP Must Be Accepted by Downstream Component.** Note that after the downstream device sends the PM\_Active\_State\_L1 DLLP it must wait for a response from the upstream component. While waiting, the downstream component must be able to accept TLPs and DLLPs from the upstream device. Although it won't send any TLPs, it must be able to send DLLPs as needed, such as ACKs for incoming TLPs. In this case, two possibilities exist:

- an ACK is returned to verify successful receipt of the TLP.
- a NAK is returned if a TLP transmission error is detected. The resulting retry of the TLP is allowed during the L1 negotiation.

**Upstream Component Receives Request to Enter L1.** The spec requires that the upstream component immediately accept or reject the request to enter the L1 state. However, it further states that prior to sending a PM\_Request\_ACK it must:

1. Block scheduling of new TLPs
2. Wait for acknowledgement of the last TLP previously sent, if necessary, and retry TLPs that receive a NAK, unless a Link Acknowledgement timeout condition occurs.

Once all outstanding TLPs have been acknowledged, and all other conditions are satisfied, the upstream device must return a PM\_Request\_ACK DLLP.

## Scenario 3: Downstream Component Receives TLP During Negotiation

During the negotiation sequence the downstream device may be instructed to send a new TLP upstream. However, a device that begins the L1 ASPM negotiation process must block new TLP scheduling. This prevents a race condition between going into L1 and sending a new TLP that would prevent entry into L1. Consequently, once the downstream device has scheduled delivery of the PM\_Request\_L1 it must complete the transition to L1 if a PM\_Request\_ACK is received. Sending a new TLP will have to wait until L1 has been entered, after which the device can initiate a transition from L1 back to L0 to send the TLP.

## Scenario 4: Upstream Component Receives TLP During Negotiation

If the upstream component needs to send a TLP or DLLP after sending the PM\_Request\_Ack, it must first complete the transition to L1. It can then initiate a change from L1 to L0 to send the packet.

# PCI Express Technology

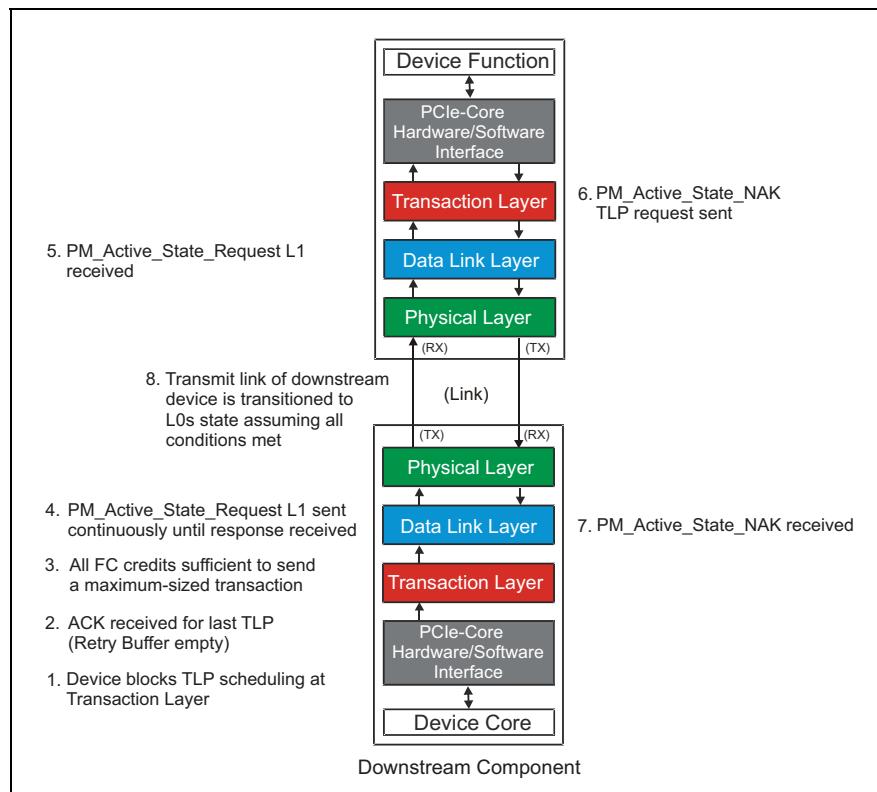
## Scenario 5: Upstream Component Rejects L1 Request

Figure 16-18 on page 752 summarizes the negotiation sequence when the upstream component rejects the request to enter the L1 ASPM state. The negotiation begins normally as the downstream component requests L1. However, the upstream device returns a PM\_Active\_State\_Nak TLP to reject the request. The reasons for rejecting the request to enter L1 include:

- L1 ASPM not supported or software has not enabled this feature
- One or more TLPs are scheduled for transfer across the Link
- ACK or NAK DLLPs are scheduled for transfer

Once the rejection message has been sent, the upstream component can continue sending TLPs and DLLPs as needed. The rejection tells the downstream component that L1 is not an option at present, and so it must transition to L0s instead, if possible.

Figure 16-18: Negotiation Sequence Resulting in Rejection to Enter L1 ASPM State



# **Chapter 16: Power Management**

---

## **Exit from L1 ASPM State**

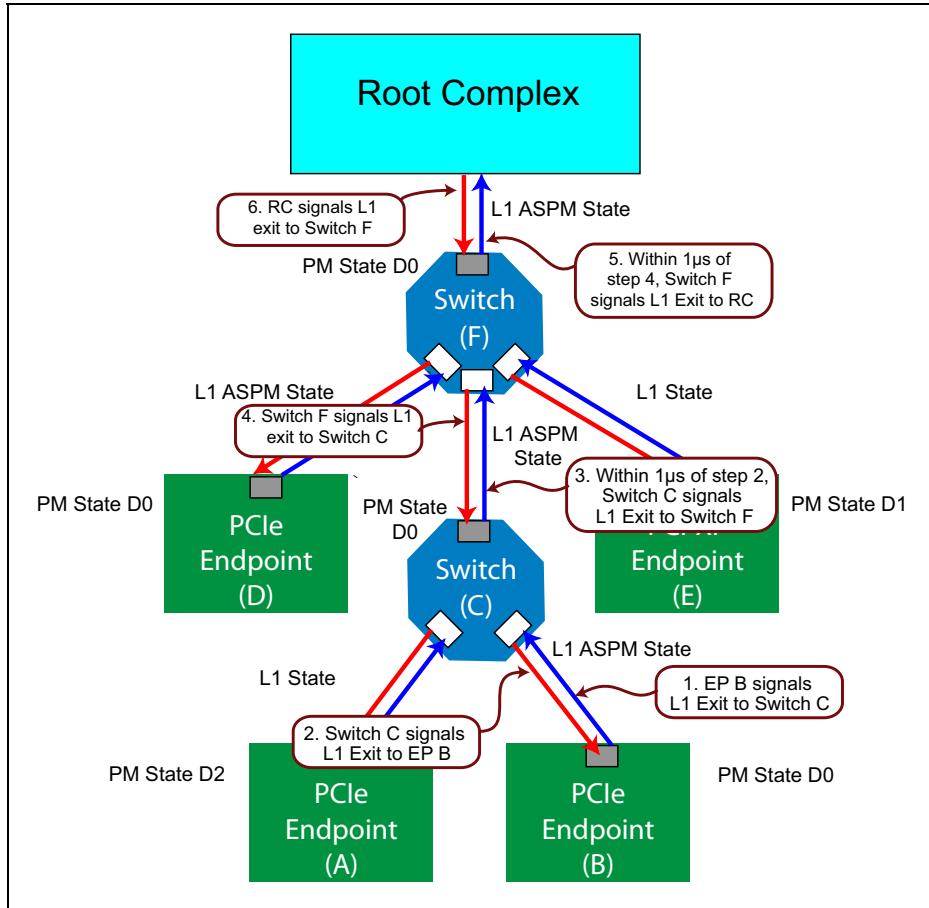
Either component can initiate the transition from L1 back to L0 when it needs to use the Link. The procedure is the same in either case and doesn't involve any negotiation. When switches are involved in exiting from L1 the spec requires that other switch ports in the ASPM low power states must also transition to the L0 state if they are in the possible path of the packet that will be sent. These issues are discussed in subsequent sections.

**L1 ASPM Exit Signaling.** The spec states that exit from L1 is invoked by exiting electrical idle, which begins by sending TS1s. The receiving port responds by sending TS1s back to the originating device and the Physical Layer follows its LTSSM protocol to complete the Recovery state and return the Link to L0. Refer to "Recovery State" on page 571 for details.

**Switch Receives L1 Exit from Downstream Component.** As pictured in Figure 16-19, the Switch must respond to L1 exit on the downstream port by returning TS1s and, within 1 $\mu$ s (from signal L1 Exit downstream), it must also exit L1 on its upstream Link if it was in that state.

# PCI Express Technology

Figure 16-19: Switch Behavior When Downstream Component Signals L1 Exit



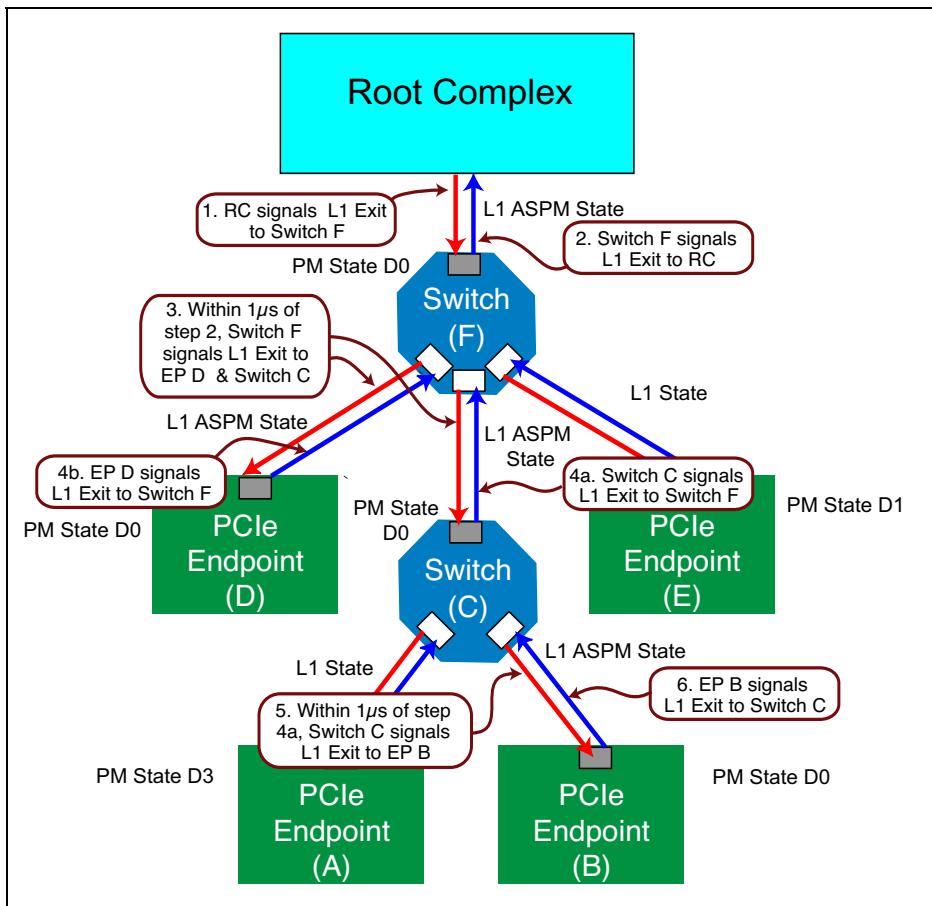
Presumably the reason the downstream component is transitioning back to L0 is because it's preparing to send a TLP upstream. Since L1 exit latencies are relatively long, a switch "must not wait until its Downstream Port Link has fully exited to L0 before initiating an L1 exit transition on its Upstream Port Link." This prevents accumulated latencies that would otherwise result if all L1 to L0 transitions occurred in a sequential fashion.

**Switch Receives L1 Exit from Upstream Component.** In this case, the switch must respond with TS1s back upstream, and within 1μs it must also send TS1s to all downstream ports that are in the L1 ASPM state to return them to L0. As in the previous example, the goal is to minimize the

# Chapter 16: Power Management

overall exit latency of returning to the L0 state for every Link in the path from the initiator to the target of the transaction. Figure 16-20 on page 755 summarizes these requirements. The Link between Switch F and EndPoint (EP) E is in the L1 state because software put EP E into the D1 state, which caused the Link to transition to L1. Only Links in the L1 ASPM state are transitioned to L0 as a result of the Root Complex (RC) initiating the exit from L1 ASPM.

Figure 16-20: Switch Behavior When Upstream Component Signals L1 Exit



## ASPM Exit Latency

PCI Express provides mechanisms to ensure that the ASPM exit latencies for L0s and L1 don't exceed the requirements of the devices. All devices report their L0s and L1 exit latencies, and Endpoints also report the total acceptable latency they can tolerate for this when performing accesses to and from the Root Complex. This acceptable latency is based on the data buffer size within the device. If the chain of devices that reside between the Endpoint and target device have a total latency that exceeds the acceptable latency reported by the Endpoint, software can disable ASPM for a given Endpoint.

The exit latencies reported by a device will change depending on whether the devices on each end of a Link share a common reference clock or not. Consequently, the Link Status register includes a bit called *Slot Clock* that specifies whether the component uses an external reference clock provided by the platform, or an independent reference clock (perhaps generated internally). Software checks these bits in devices at both ends of each Link to determine whether they both use it and thus share a common clock. If so, software sets the *Common Clock* bit to report this in both devices. Figure 16-21 on page 757 illustrates the registers and related bit fields involved in managing the ASPM exit latency.

### Reporting a Valid ASPM Exit Latency

Because the clock configuration affects the exit latency that a device will experience, devices must report the source of their reference clock via the *Slot Clock* status bit within the Link Status register. This bit is initialized by the component to report the source of its reference clock. If this bit is set to 1, the clock uses the platform generated reference clock and if it's cleared (0) an independent clock is used.

If system firmware or software determines that both components on the Link use the platform clock then the reference clocks within both devices will be in phase. This results in shorter exit latencies from L0s and L1, and is reported in the *Common Clock* field of the Link Control register. Components must then update their reported exit latencies to reflect the correct value. Note that if the clocks are not common then the default values will be correct and no further action is required.

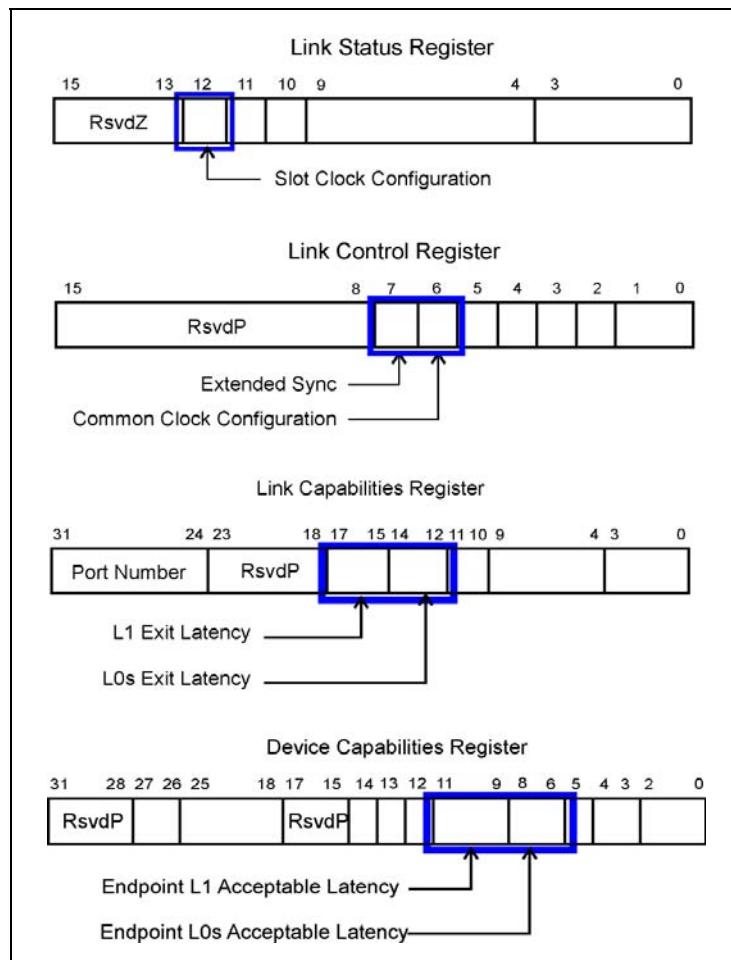
**L0s Exit Latency Update.** Exit latency for L0s is reported in the Link Capability register based on the default assumption that a common clock implementation does not exist. L0s exit latency is also reported in the TS1s

# Chapter 16: Power Management

used during Link training as the number of FTS Ordered Sets (N\_FTS) required to exit L0s. If software then detects a common clock implementation, it sets the Common Clock field writes to the *Retrain Link* bit in the Link Control register to force Link training to repeat. During retraining new N\_FTS values are reported and in the *L0s Latency* field of the Link Capability register.

**L1 Exit Latency Update.** Following Link retraining, new values will also be reported in the *L1 Latency* field.

Figure 16-21: Config. Registers for ASPM Exit Latency Management and Reporting



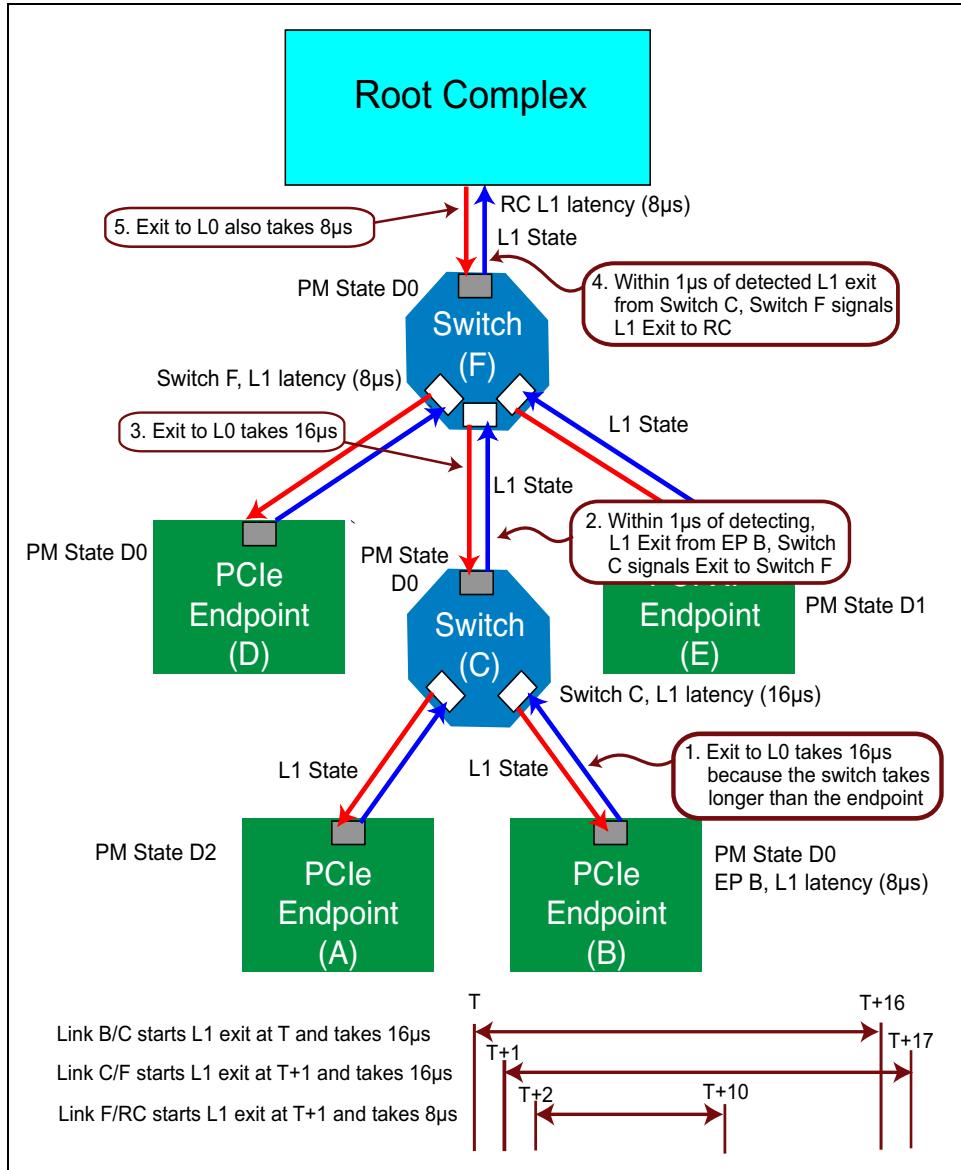
## Calculating Latency from Endpoint to Root Complex

Figure 16-22 on page 759 illustrates an Endpoint whose transactions must transverse two switches to reach the Root Complex. Presuming that all Links in the path are in the L1 state, let's take the example that Endpoint B needs to send a packet to main memory.

1. First, it begins the wake sequence by initiating a TS1 ordered set on its Link at time "T." The L1 exit latency for EP B is a maximum of 8 $\mu$ s, but Switch C has a maximum exit latency of 16 $\mu$ s. Therefore, the exit latency for this Link is 16 $\mu$ s.
2. Within 1 $\mu$ s of detecting the L1 exit on Link B/C, Switch C signals L1 exit on Link C/F at T+1 $\mu$ s.
3. Link C/F completes its exit from L1 in 16 $\mu$ s, at T+17 $\mu$ s.
4. Switch F signals an exit from L1 to the Root Complex within 1 $\mu$ s of detecting L1 exit from Switch C (T+2 $\mu$ s).
5. Link F/RC completes exit from L1 in 8 $\mu$ s, completing at T+10 $\mu$ s.
6. Total latency to transition path to target back to L0 = T+17 $\mu$ s.

# Chapter 16: Power Management

Figure 16-22: Example of Total L1 Latency



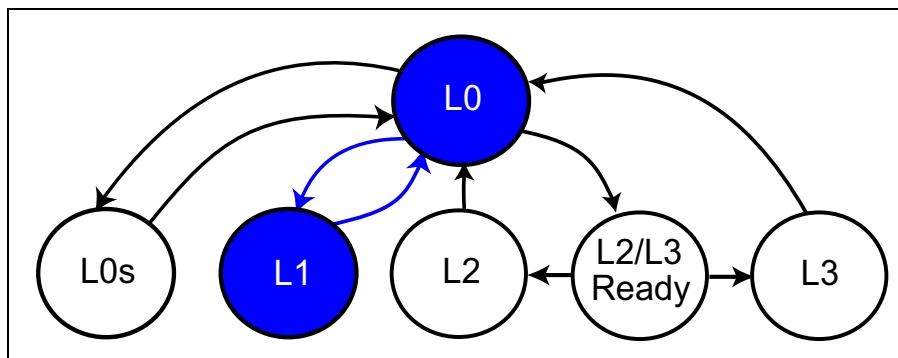
## Software Initiated Link Power Management

When software initiates configuration writes to change the power state for power conservation, devices must respond by transitioning their Link to the corresponding low power state.

### D1/D2/D3<sub>Hot</sub> and the L1 State

The spec requires that when all Functions within a device have been placed into any of the low power states (D1, D2, or D3<sub>hot</sub>), the device must initiate a transition to the L1 state as shown in Figure 16-23. A device returns to L0 as a result of software initiating a configuration access to the device or a device initiated event.

Figure 16-23: Devices Transition to L1 When Software Changes their Power Level from D0



Upon receiving a configuration write to the *Power State* field of the PMCSR register, a device initiates the change from L0 to L1 by sending a PM\_Enter\_L1 DLLP to the upstream component.

### Entering the L1 State

The procedure to place the Link into an L1 state is illustrated in Figure 16-24 on page 762. The steps in the figure are described in greater detail in the following list:

1. Once a device recognizes that all its Functions are in the D2 state, it must prepare to transition the Link into L1. This begins with blocking new TLPs from being scheduled.

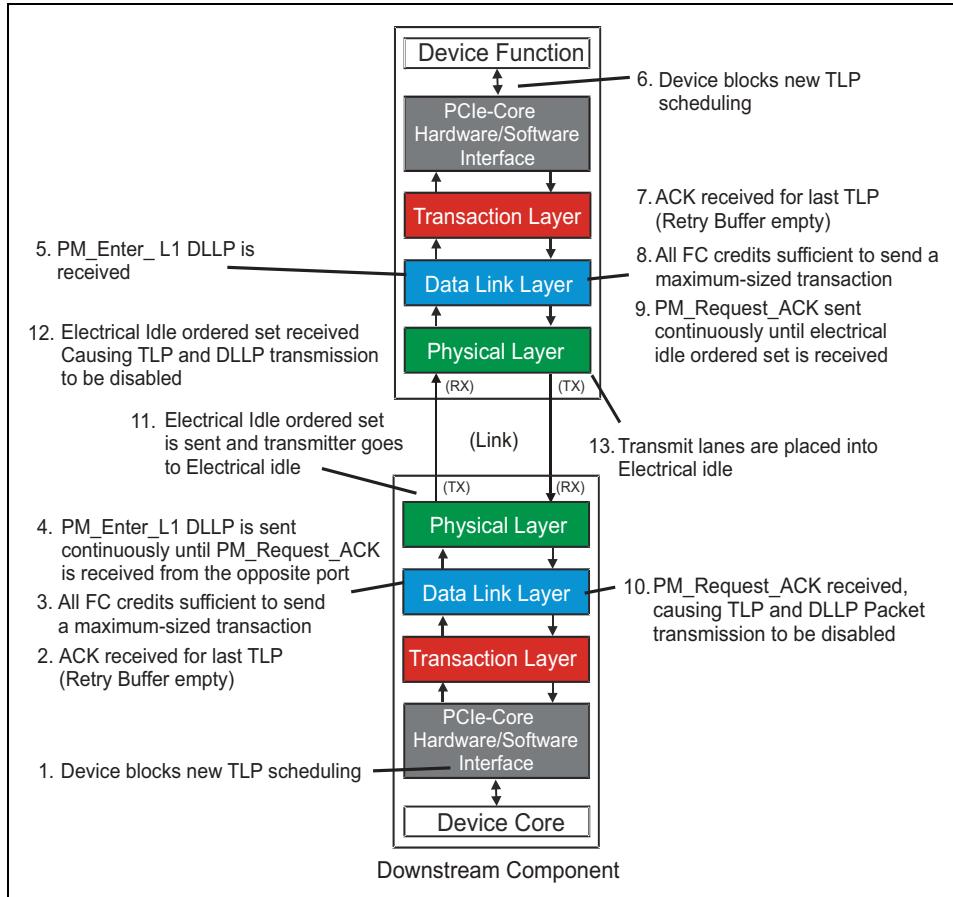
## **Chapter 16: Power Management**

---

2. A TLP from the downstream Endpoint may not have been acknowledged prior to receiving the request to enter D2. The device must not respond to a request to change the Link power until all outstanding TLPs have been acknowledged. In other words, the Replay Buffer must be empty before proceeding to the L1 state.
3. Because of the long latencies involved in returning the Link to its active state, a device must be able to send a maximum-sized TLP immediately upon return to the active state. Since a lack of Flow Control credits could block this, the Endpoint must have sufficient credits to permit transmission of the biggest packet supported for each Flow Control type before entering L1.
4. When the requirements listed above have been met, the Endpoint sends a PM\_Enter\_L1 DLLP to the upstream device. This instructs the upstream component to put the Link into L1. The PM\_Enter\_L1 is repeated until a PM\_Request\_ACK DLLP is received from the upstream device.
5. When the upstream component receives PM\_Enter\_L1, it begins its preparation by performing steps 6, 7, and 8. This is the same preparation as performed by the downstream component prior to signaling the L1 transition.
6. All new TLP scheduling is blocked.
7. In the event that a previous TLP has not yet been acknowledged, the upstream device will wait until all transactions in the Replay Buffer have been acknowledged.
8. Sufficient Flow Control credits must be accumulated to ensure that the largest TLP can be transmitted for each Flow Control type.
9. The upstream component sends a PM\_Request\_ACK DLLP to confirm that it's ready to enter the L1 state. This DLLP is repeated until an Electrical Idle ordered set is received, indicating that it's been accepted.
10. When the downstream component receives the acknowledgement, it sends an EIOS and places its transmit lanes into electrical idle (transmitter is in Hi-Z state).
11. The upstream component recognizes the EIOS and places its transmit lanes into electrical idle. The Link has now entered the L1 state.

# PCI Express Technology

Figure 16-24: Procedure Used to Transition a Link from the L0 to L1 State



## Exiting the L1 State

An exit from the L1 state can be initiated by either the upstream or downstream component, as discussed below. This section also summarizes the signaling protocol used to exit L1.

**Upstream Component Initiates.** Software may need to use a device which is currently in a low-power state, and that means the Power Management software must issue a configuration write to change its power state back to D0. When the configuration Request is ready to be sent from the upstream component (a Root Port or downstream Switch Port) the port will exit the electrical idle state and initiate re-training to return the Link to the

# Chapter 16: Power Management

---

L0 state. Once the Link is active, the configuration write can be delivered to the device to transition it back to D0, at which point it's ready for normal use.

**Downstream Component Initiates L1 to L0 Transition.** In the L1 state the reference clock and power are still applied to devices on the Link. That allows a downstream device to be designed to monitor external events and trigger a Power Management Event (PME) when it occurs. In conventional PCI, this is reported by a side-band PME# signal, and system board logic usually uses it to generate an interrupt that informs the CPU of the need to bring the device back to full operation. PCIe eliminates the side-band signal and instead sends an in-band message to report the PME (see "The PME Message" on page 769 for details).

**The L1 Exit Protocol.** In the L1 state both directions of the Link are in the electrical idle state. A device signals an exit from L1 by changing from electrical idle and sending TS1s. When the Link neighbor detects the exit from electrical idle it sends TS1s back. This sequence triggers both devices to enter the Recovery state and, when that has completed its operation, both devices will have returned to the L0 state.

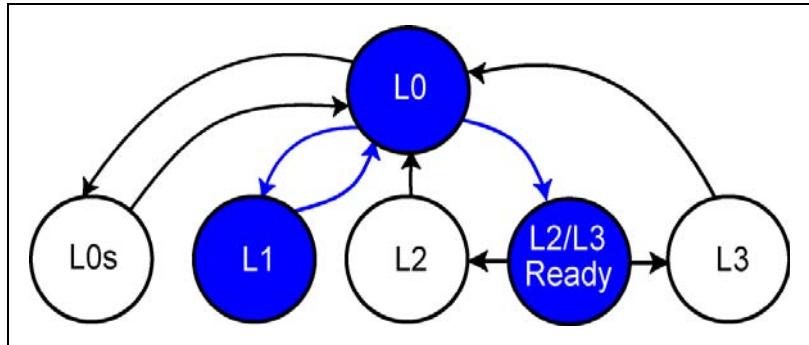
---

## L2/L3 Ready — Removing Power from the Link

Once software has placed all Functions within a Device into the D3<sub>hot</sub> state power can be safely removed from the device. A typical application for this would be to place all devices in the system into D3 and then remove power from them all to achieve the lowest power consumption. However, the spec does not give details of the actual mechanism that would be used to remove clock and power or require that a particular sequence be followed, allowing for a variety of implementations.

The state transitions to prepare devices for power removal involve the preliminary steps of entering L1 and then returning to L0 before arriving at the L2/L3 Ready state as illustrated in Figure 16-25 on page 764.

Figure 16-25: Link States Transitions Associated with Preparing Devices for Removal of the Reference Clock and Power



## L2/L3 Ready Handshake Sequence

The spec does require a handshake sequence when transitioning to the L2/L3 Ready state. This ensures that all devices are ready for reference clock and power removal, and also that inband PME messages being sent to the Root Complex won't accidentally be lost when power is removed.

Consider the following example of the handshake sequence required for removing the reference clock and power from PCIe devices in the fabric. This example assumes a system-wide power down is being initiated, but the sequence can also apply to individual devices. The steps are summarized below and shown in Figure 16-26 on page 766. The overall sequence is represented in two parts labeled A and B. The Link state transitions involved in the complete sequence include:

- L0 --> L1 (when software places a device into D3)
- L1 --> L0 (when software initiates a PME\_Turn\_Off message)
- L0 --> L2/L3 Ready (resulting from the completion of the PME\_Turn\_Off handshake sequence, which culminates in a PM\_Enter\_L23 DLLP being sent by the device and the Link going to electrical idle)

The following steps detail the sequence illustrated in Figure 16-26 on page 766.

1. Power Management software first places all Functions in the PCIe fabric into their D3 state.
2. All devices transition their Links to the L1 state when they enter D3.
3. Power Management software initiates a PME\_Turn\_Off TLP message,

## **Chapter 16: Power Management**

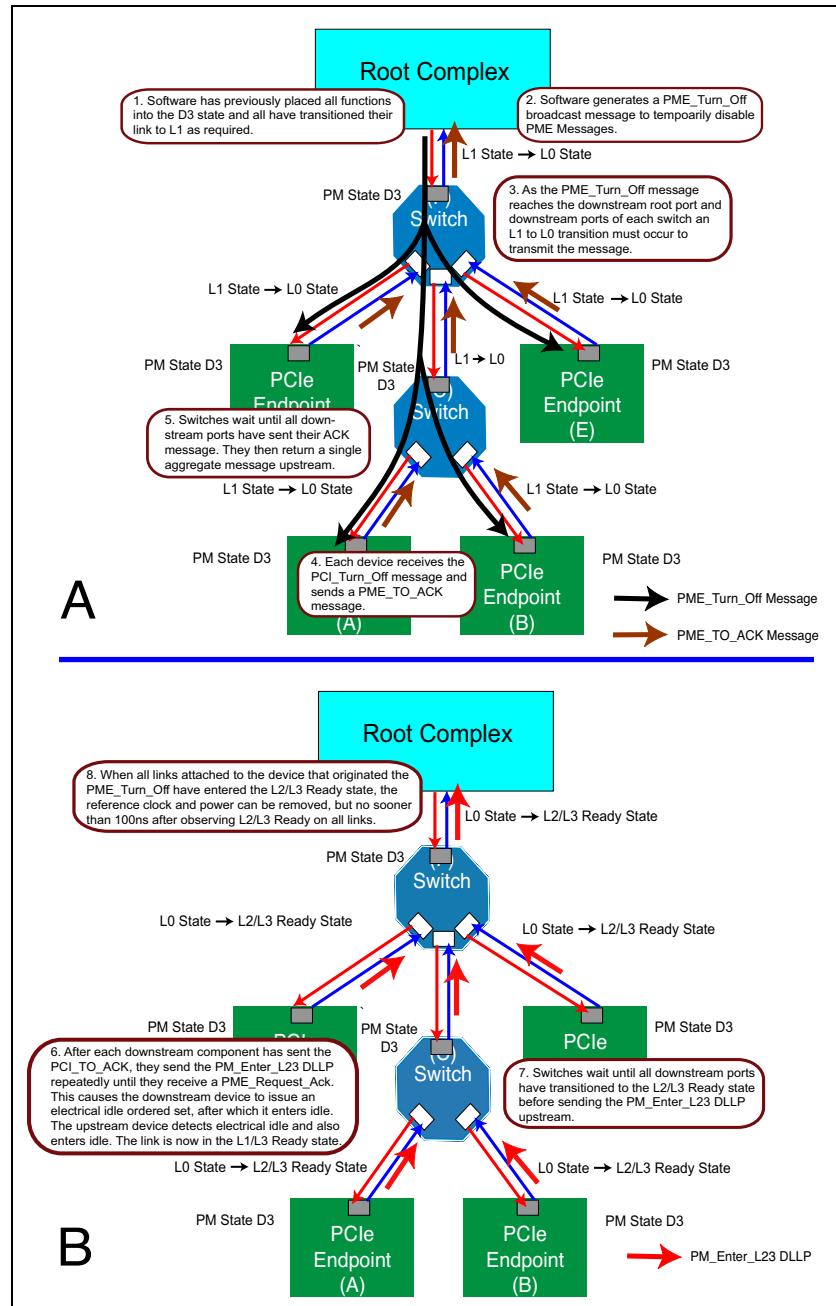
---

which is broadcast from all Root Complex ports to all devices. This prevents PME Messages from being lost in case they were in progress upstream when power was removed. Note that delivery of this TLP causes each Link to transition back to L0 so it can be forwarded downstream.

4. All devices must receive and acknowledge the PME\_Turn\_Off message by returning a PME\_TO\_ACK TLP message while in the D3 state.
5. Switches collect the PME\_TO\_ACK messages from all of their enabled downstream ports and forward just one aggregated PME\_TO\_ACK message upstream toward the Root Complex. That's because these messages have the routing attribute set as "Gather and Route to the Root".
6. After sending the PME\_TO\_ACK, when it is ready to have the reference clock and power removed, devices send a PM\_Enter\_L23 DLLP repeatedly until a PM\_Request\_ACK DLLP is returned. The Links that enter the L2/L3 Ready state last are those attached to the device originating the PME\_Turn\_Off message (the Root Complex in this example).
7. The reference clock and power can finally be removed when all Links have transitioned to the L2/L3 state, but not sooner than 100ns after that. If auxiliary power ( $V_{AUX}$ ) is supplied to the devices, the Link transitions to L2. If no AUX power is available the Links will be in the L3 state.

# PCI Express Technology

Figure 16-26: Negotiation for Entering L2/L3 Ready State



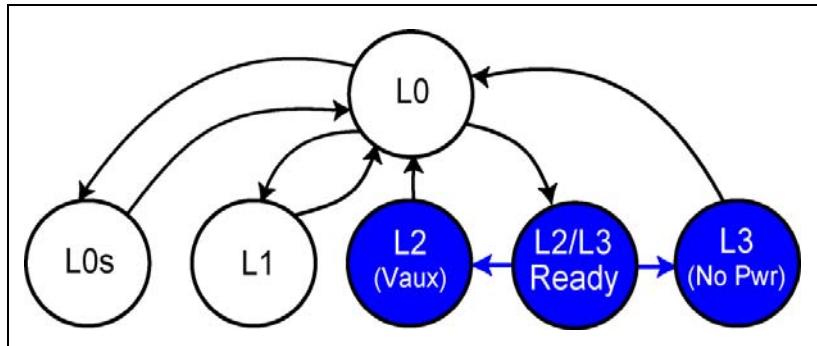
# Chapter 16: Power Management

## Exiting the L2/L3 Ready State — Clock and Power Removed

As illustrated in the state diagram in Figure 16-27, a device exits the L2/L3 Ready state when power is removed and has only two choices. When  $V_{AUX}$  is available the transition is to L2, otherwise the transition is to L3.

Link state transitions are normally controlled by the LTSSM in the Physical Layer. However, transitions to L2 and L3 result from main power being removed and the LTSSM is not operational then. Consequently, the spec refers to L2 and L3 as pseudo-states defined for explaining the resulting condition of a device when power is removed.

Figure 16-27: State Transitions from L2/L3 Ready When Power is Removed



### The L2 State

Some devices are designed to monitor external events and initiate a wakeup sequence to restore power to handle them. Since main power is removed, these device will need a power source like  $V_{AUX}$  to be able to monitor the events and to signal a wakeup.

### The L3 State

In this state the device has no power and therefore no means of communication. Recovery from this state requires the system to restore power and the reference clock. That causes devices to experience a fundamental reset, after which they'll need be initialized by software to return to normal operation.

## Link Wake Protocol and PME Generation

The wake protocol provides a method for an Endpoint to reactivate the upstream Link and request that software return it to D0 so it can perform required operations. PCIe PM is designed to be compatible with PCI-PM software, although the methods are different.

Rather than using a sideband signal, PCIe devices use an inband PME message to notify PM software of the need to return the device to D0. The ability to generate PME messages may optionally be supported in any of the low power states. Recall that a device reports which PM states it supports for PME message delivery.

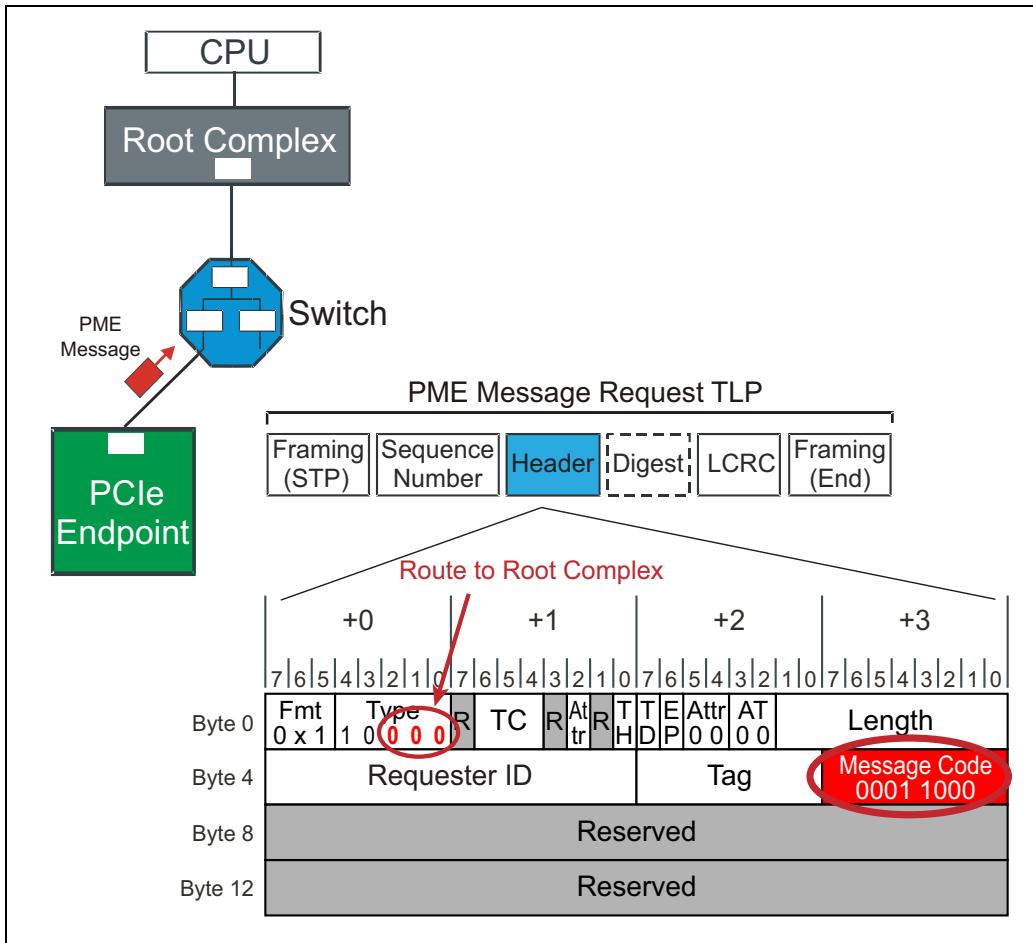
PME messages can only be delivered when the Link state is L0. The latency involved in reactivating the Link is based on a device's PM and Link state, but can include the following:

1. Link is in non-communicating (L2) state — when a Link is in the L2 state it cannot communicate because the reference clock and main power have been removed. No PME message can be sent until clock and power are restored, a Fundamental Reset is asserted, and the Link is re-trained. These events will be triggered when a device signals a wakeup. This may result in all Links being re-awakened in the path between the device needing to communicate and the Root Complex.
2. Link is in communicating (L1) state — when a Link is in the L1 state clock and main power are still active; thus, a device simply exits the L1 state, goes to the Recovery state to re-train the Link, and returns the Link to L0. Once the Link is in L0 the PME message is delivered. Note that the devices never send a PME message while in the L2/L3 Ready state because entry into that state only occurs after PME notification has been turned off, in preparation for clock and power to be removed. (See “L2/L3 Ready Handshake Sequence” on page 764.)
3. PME is delivered (L0) — If the Link is in the L0 state, the device transfers the PME message to the Root Complex, notifying Power Management software that the device has observed an event that requires the device be placed back into its D0 state. Note that the message contains the Requester ID (Bus#, Device#, and Function#) of the device. This quickly informs software which device needs service.

## The PME Message

The PME message is delivered by devices that support PME notification. The message format is illustrated in Table 16-28 on page 769. The message may be initiated by a device in a low power state (D1, D2, D3<sub>hot</sub>, and D3<sub>cold</sub>) and is sent immediately upon return of the Link to L0.

Figure 16-28: PME Message Format



# PCI Express Technology

---

The PME message is a Transaction Layer Packet that has the following characteristics:

- TC and VC are zero (no QoS applies)
- Routed implicitly to the Root Complex
- Handled as Posted Transaction
- Relaxed Ordering is not permitted, forcing all transactions in the fabric between the signaling device and the Root Complex to be delivered to the Root Complex ahead of the PME message

---

## The PME Sequence

Devices may support PME in any of the low power states as specified in the PM Capabilities register. This register also specifies the amount of  $V_{AUX}$  current used by the device if it supports wakeup in the D3<sub>cold</sub> state. The basic sequence of events associated with sending a PME to software is specified below and presumes that the device and system are enabled to generate PME and the Link has already been transitioned to the L0 state:

1. The device issues the PME message on its upstream port.
2. PME messages are implicitly routed to the Root Complex. Switches in the path transition their upstream ports to L0 if necessary and forward the packet upstream.
3. A root port receives the PME and forwards it to the Power Management Controller.
4. The controller informs power management software, typically with an interrupt. Software uses the Requester ID in the message to read and clear the PME\_Status bit in the PMCSR and return the device to the D0 state. Depending on the degree of power conservation, the PCI Express driver may also need to restore the devices configuration registers.
5. PM Software may also call the device driver in the event that device context was lost as a result of being placed in a low power state. If so, device software restores information within the device.

---

## PME Message Back Pressure Deadlock Avoidance

### Background

The Root Complex typically stores the PME messages it receives in a queue, and calls PM software to handle each one. A PME is held in this queue until PM soft-

# **Chapter 16: Power Management**

---

ware reads the PME\_Status bit from the requesting device's PMCSR register. Once the configuration read transaction completes, this PME message can be removed from the internal queue.

## **The Problem**

Deadlock can occur if the following scenario develops:

1. Incoming PME Messages have filled the PME message queue but other PME messages have been issued downstream from the same root port.
2. PM software initiates a configuration read request from the Root to read PME\_Status from the oldest PME requester.
3. The corresponding split completion must push all previously posted PME messages ahead of it based on transaction ordering rules.
4. The Root Complex cannot accept a new PME message because the queue is full, so the path is temporarily blocked. But that also means that the read completion can't reach the Root Complex to clear the older entry in the queue.
5. No progress can be made and deadlock occurs.

## **The Solution**

The problem is avoided if the Root Complex always accepts new PME messages, even when they would overflow the queue. In this case, the Root simply discards the later PME messages. To prevent a discarded PME message from being lost permanently, a device that sends a PME message is required to measure a time-out interval, called the PME Service Time-out. If the device's PME\_Status bit is not cleared with 100 ms (+ 50%/- 5%), it assumes its message must have been lost and it re-issues the message.

---

## **The PME Context**

Devices that generate PME must continue to power portions of the device that are used for detecting, signaling, and handling PME events, referred to collectively as the PME context. Devices that support PME in the D<sub>3,cold</sub> state use auxiliary power to maintain the PME context when the main power is removed. Items that are typically part of the PME context include:

- PME\_Status bit (required) — set when a device sends a PME message and cleared by PM software. Devices that support PME in the D<sub>3,cold</sub> state must implement the PME\_Status bit as "sticky," meaning that the value survives a fundamental reset.

# PCI Express Technology

---

- PME\_Enable bit (required) — this bit must remain set to continue enabling a Function's ability to generate PME messages and signal wakeup. Devices that support PME in the D3<sub>cold</sub> state must implement PME\_Enable as "sticky," meaning that the value survives a fundamental reset.
- Device-specific status information — for example, a device might preserve event status information in cases where several different types of events can trigger a PME.
- Application-specific information — for example, modems that initiate wakeup would preserve Caller ID information if supported.

---

## Waking Non-Communicating Links

When a device that supports PME in the D3cold state needs to send a PME message, it must first transition the Link to L0. This is sometimes referred to as a wakeup. PCI Express defines two methods of triggering the wakeup of non-communicating Links:

- Beacon — an in-band indicator driven by AUX power
- WAKE# Signal — a sideband signal driven by AUX power

In both cases, PM software must be notified to restore main power and the reference clock. This also causes a fundamental reset that forces a device into the D0<sub>uninitialized</sub> state. Once the Link transitions to L0, the device sends the PME message. Since a reset is required to re-activate the Link, devices must maintain PME context across the reset sequence described above.

### Beacon

This signaling mechanism is designed to operate on AUX power and doesn't require much power. The beacon is simply a way of notifying the upstream component that software should be notified of the wakeup request. When switches receive a beacon on a downstream port, they in turn signal beacon on their upstream port. Ultimately, the beacon reaches the root complex, where it generates an interrupt that calls PM software.

Some form-factors require beacon support for waking the system while others don't. The spec requires compliance with the form-factor specs, and doesn't require beacon support for devices if their form-factor doesn't. However, for "universal" components designed for use in a variety of form-factors, beacon support is required. See "Beacon Signaling" on page 483 for details.

# **Chapter 16: Power Management**

---

## **WAKE#**

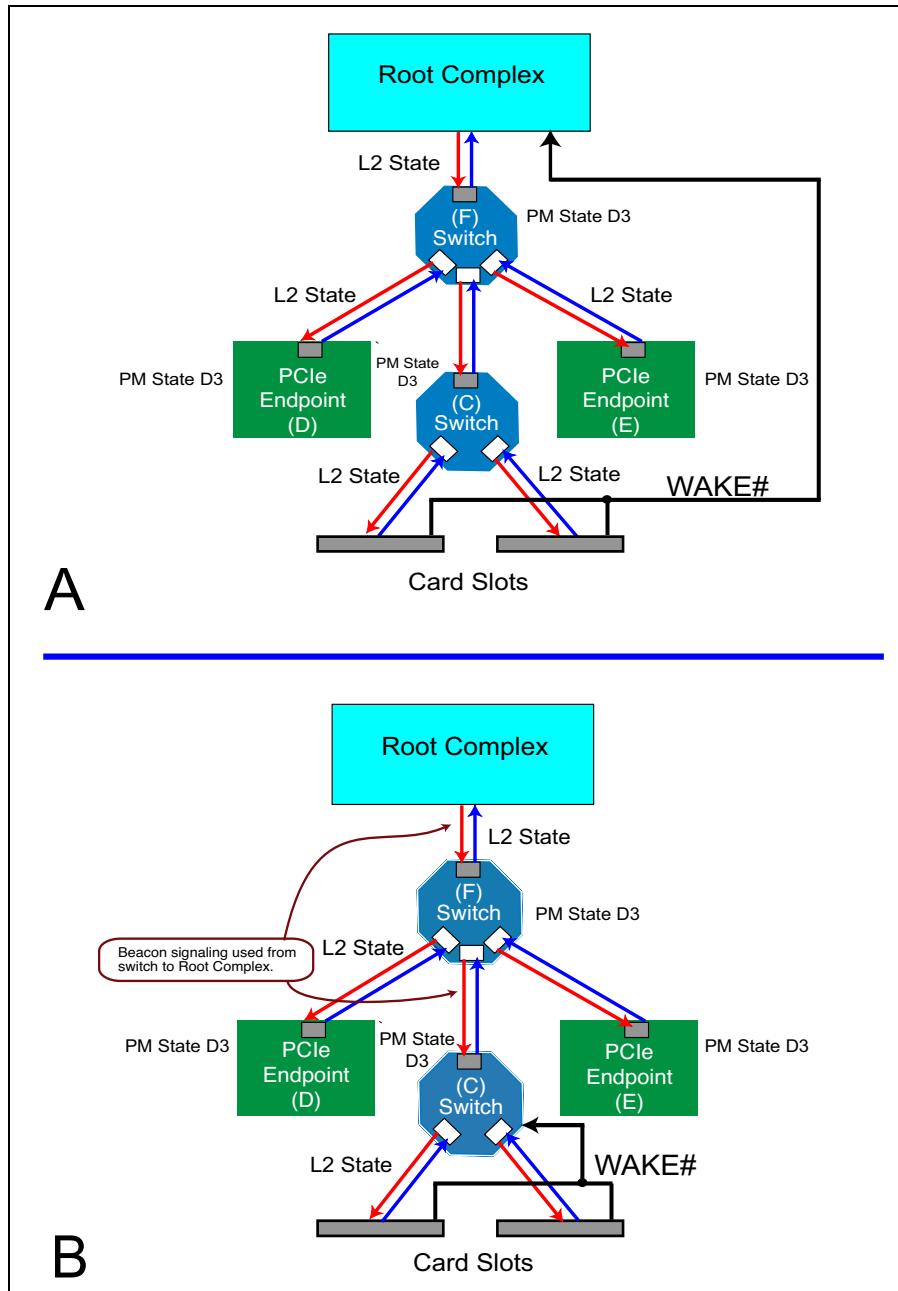
PCI Express provides a sideband signal called WAKE# as an alternative to the beacon that can be routed directly to the Root or to other system logic to notify PM software. In spite of the desire to minimize the pin count of a Link, the motivation for adding this extra pin is easy to understand. The reason is that a component must consume auxiliary power to be able to recognize a beacon on a downstream port and then forward it to an upstream port. In a battery-powered system auxiliary power is jealously guarded because it drains the battery even when the system isn't doing any work. The preferred solution in that case would be to bypass as many components as possible when delivering the wakeup notification, and the WAKE# pin serves that purpose very well. On the other hand, if power is not a concern then the WAKE# pin might be considered less desirable.

A hybrid implementation may also be used. In this case, WAKE# is sent to a switch, which in turn sends the beacon on its upstream port. The options are illustrated in Figure 16-29 on page 774 A and B. Note that when asserted, the WAKE# signal remains low until the PME\_Status bit is cleared by software.

This signal must be implemented by ATX or ATX-based connectors and cards as well as by the mini-card form factor. No requirement is specified for embedded devices to use the WAKE# signal.

# PCI Express Technology

Figure 16-29: WAKE# Signal Implementations

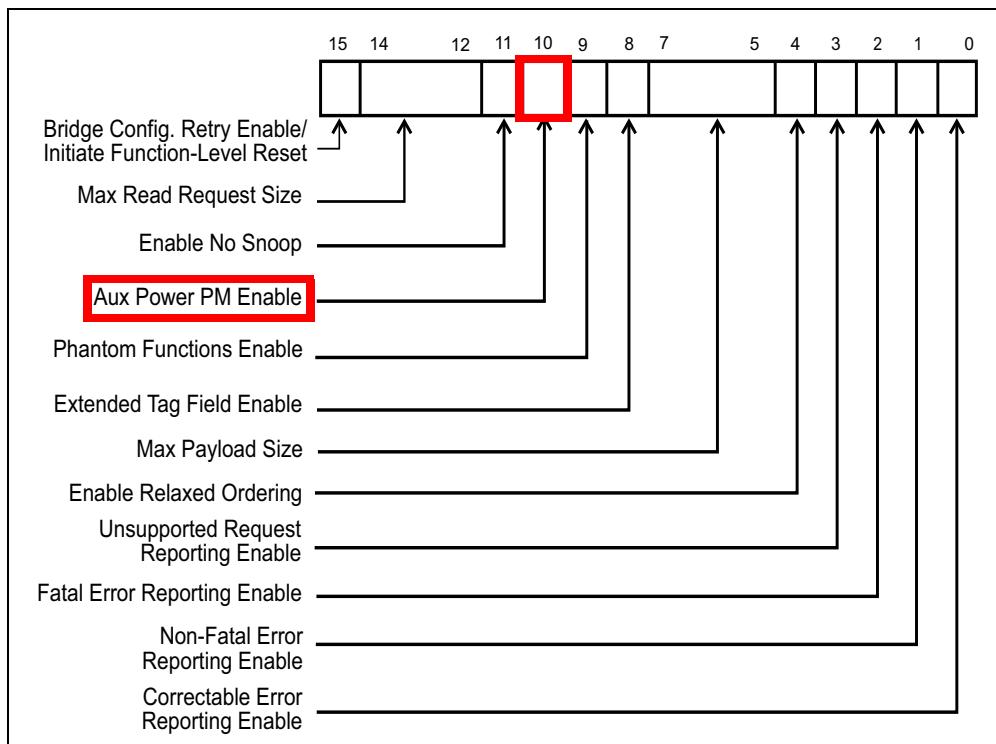


## Auxiliary Power

Devices that support PME in the D<sub>3,cold</sub> state must support the wakeup sequence and are allowed by the PCI-PM spec to consume the maximum auxiliary current of 375 mA (otherwise only 20mA). The amount of current they need is reported in the *Aux\_Current* field of the PM Capability registers. Auxiliary power is enabled when the *PME\_Enable* bit is set within the PMCSR register.

PCI Express extends the use of auxiliary power beyond the limitations given by PCI-PM. Now, any Device may consume the maximum auxiliary current if enabled by setting the *Aux Power PM Enable* bit of the Device Control register, illustrated in Figure 16-30 on page 775. This gives devices the opportunity to support other things like SM Bus while in a low power state. As in PCI-PM the amount of current consumed by a device is reported in the *Aux\_Current* field in the PMC register.

Figure 16-30: Auxiliary Current Enable for Devices Not Supporting PMEs



---

## Improving PM Efficiency

---

### Background

As processors and other system components acquire better power management mechanisms, peripherals like PCIe components start to appear as a bigger contributor to power consumption in PC systems. Earlier generations of PCIe allowed some software and hardware power management, but coordinating PM decisions with the system was not a high priority and consequently software visibility and control was limited.

One problem that can arise from this lack of coordination happens when the system goes into a sleep state but the devices remain operational. Such devices can initiate interrupts or DMA traffic that would require the system to wake up to handle them, even though they were low-priority events, and thus defeat the goal of power conservation.

It can also happen that the system is unaware of how long the devices can afford to wait from the time they request system service (like a memory read) until they get a response. Without that information, software is often forced to assume that the response time must always be minimal and therefore power management policies can't afford enough time to do much. However, if the system was aware of time windows when a fast response was not needed, it could be more aggressive with power management and stay in a low power state for a longer time without risking performance problems. The 2.1 spec revision added two new features to address these problems.

---

### OBFF (Optimized Buffer Flush and Fill)

The first of these mechanisms is Optimized Buffer Flush and Fill, which provides a mechanism for Endpoints to be made aware of the system power state and therefore the best times to do data transfers to and from the system.

#### The Problem

The problem with bus-master capable devices is that if they're not aware of the system power status, they may initiate transactions at times when it would be better to wait. The diagram in Figure 16-31 on page 777 illustrates the problem in simple terms: there are many components initiating events and as a result,

# Chapter 16: Power Management

the times without activity when the system is idle and can go to sleep are few and short-lived. In contrast, Figure 16-32 on page 777 illustrates an improvement in which the same events are grouped and serviced together so that the times when the system is idle enough to go to sleep are both more frequent and of longer duration. Clearly, this would result in better power conservation and fortunately, it's not difficult to implement. PCIe components simply need to understand what they should do based on the system power state, and they'll need a way to learn what that state currently is.

Figure 16-31: Poor System Idle Time

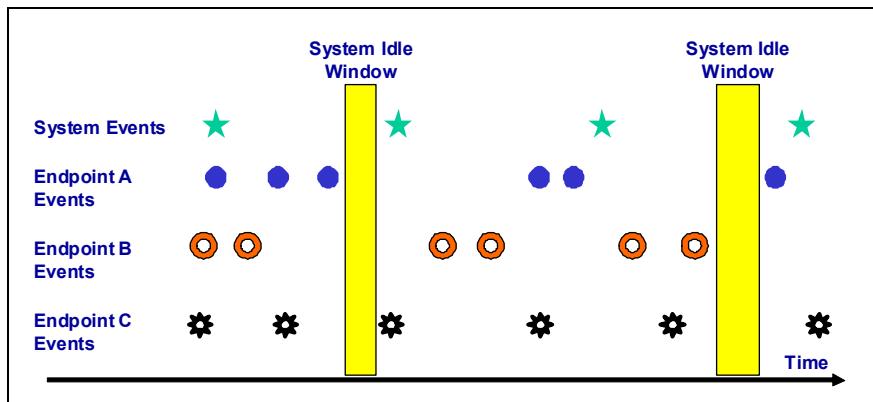
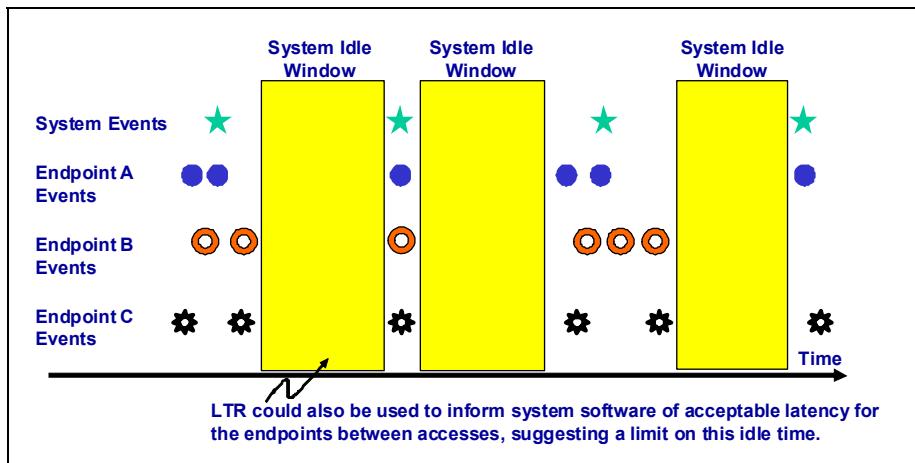


Figure 16-32: Improved System Idle Time

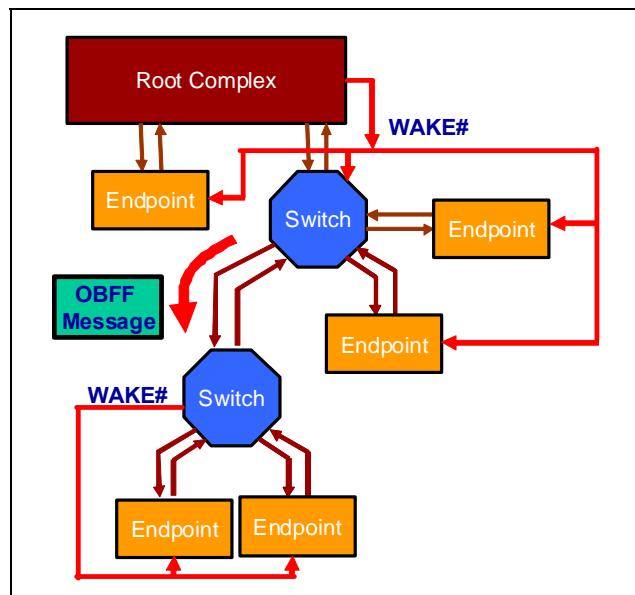


## The Solution

OBFF is an optional hint that a system can use to inform components about optimal time windows for traffic. It's just a hint, though, so bus-master-capable devices can still initiate traffic whenever they like. Of course, power consumption will be negatively affected if they do, so overriding the OBFF hints should be avoided as much as possible. The information is communicated in one of two ways: by sending messages to the Endpoints or by toggling the WAKE# pin. If both options are available, using the pin is strongly recommended because it avoids the counter-productive step of using excess power, possibly across several Links, to inform a component about the current system power state. In fact, the OBFF message should only be used if the WAKE# pin is not available.

Figure 16-33 on page 778 gives an example showing a mix of both communication types. Using the pin is required if it's available, but in this example it's not an option between the two switches. To work around this problem, the upper switch can translate the state received on the WAKE# pin into a message going downstream. It should perhaps be noted here that switches are strongly encouraged to forward all OBFF indications downstream but not required to do so. It may be necessary, especially when using messages, to discard or collapse some indications and that is permitted.

Figure 16-33: OBFF Signaling Example

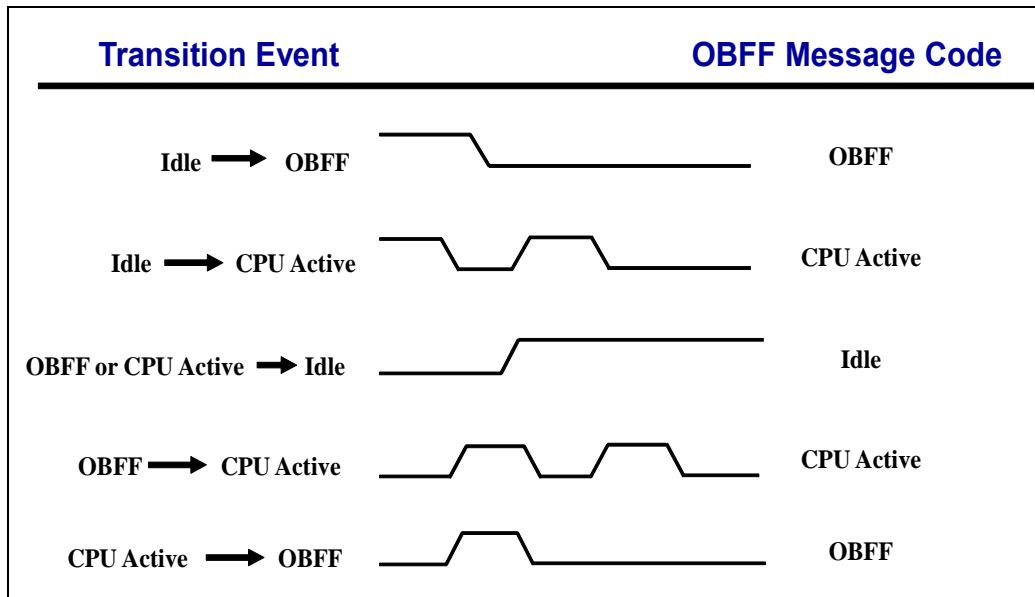


# Chapter 16: Power Management

**Using the WAKE# Pin.** This pin, previously only used to inform the system that a component needed to have power restored, is given an extra meaning as the simplest and lowest-power option for communicating system power status to PCIe components. It's optional, and the protocol is fairly simple: the WAKE# pin toggles to communicate the system state. As seen in Figure 16-34 on page 779, there are several transitions but only three states, which are described below:

1. CPU Active - system awake; all transactions OK. This is every component's initial state.
2. OBFF - system memory path available; transfers to and from memory are OK, but other transactions should wait for a higher power state.
3. Idle - wait for a higher state before initiating.

Figure 16-34: WAKE# Pin OBFF Signaling



When the CPU Active or OBFF state is indicated, it's recommended that the platform not return to the Idle state for at least 10 µs so as to give components enough time to deliver the packets they may have been queuing up while in the previous Idle state. However, since that timing isn't required, it's also recommended that Endpoints not assume they'll have a certain amount of time in a CPU Active or OBFF window. Along the same lines, the platform is allowed to indicate that it's going to Idle before it actually does

# PCI Express Technology

---

so as to give components advance notice that it's time to finish. The case this early notice is specifically designed to avoid is having an Endpoint start a transfer just as the platform goes to Idle, causing an immediate exit from the Idle state. The spec strongly recommends that this should be the only reason for an early indication of the Idle state and also that this advance notice time should be as short as possible.

Interestingly, the WAKE# pin can still be used for its original purpose of allowing a component to wake the system, and it's no surprise that this might confuse other components that are monitoring that pin for OBFF information. That could result in sub-optimal behavior in power or performance, but this is considered a recoverable situation so no steps were taken to guard against it. To cover all of these cases, any time the signal is unclear the default state will be CPU Active.

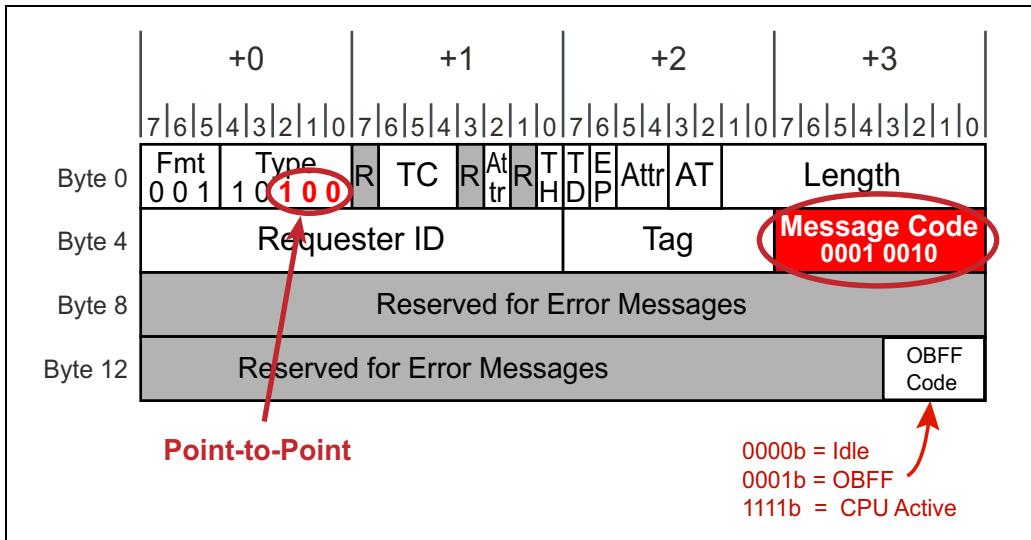
**Using the OBFF Message.** As mentioned earlier, OBFF information can be communicated using a message, although it's recommended that this only be used if the WAKE# pin is not available. These messages only flow downstream from the Root. The message contents are shown in Figure 16-35 on page 781, including the Routing type 100b (point-to-point) and an OBFF Code that gives the following values (all other codes are reserved):

1. 1111b - CPU Active
2. 0001b - OBFF
3. 0000b - Idle

If a reserved code is received, components must treat it as "CPU Active." If a Port receives an OBFF message but doesn't support OBFF or hasn't enabled it yet, it must treat it as an Unsupported Request (Completion status UR).

# Chapter 16: Power Management

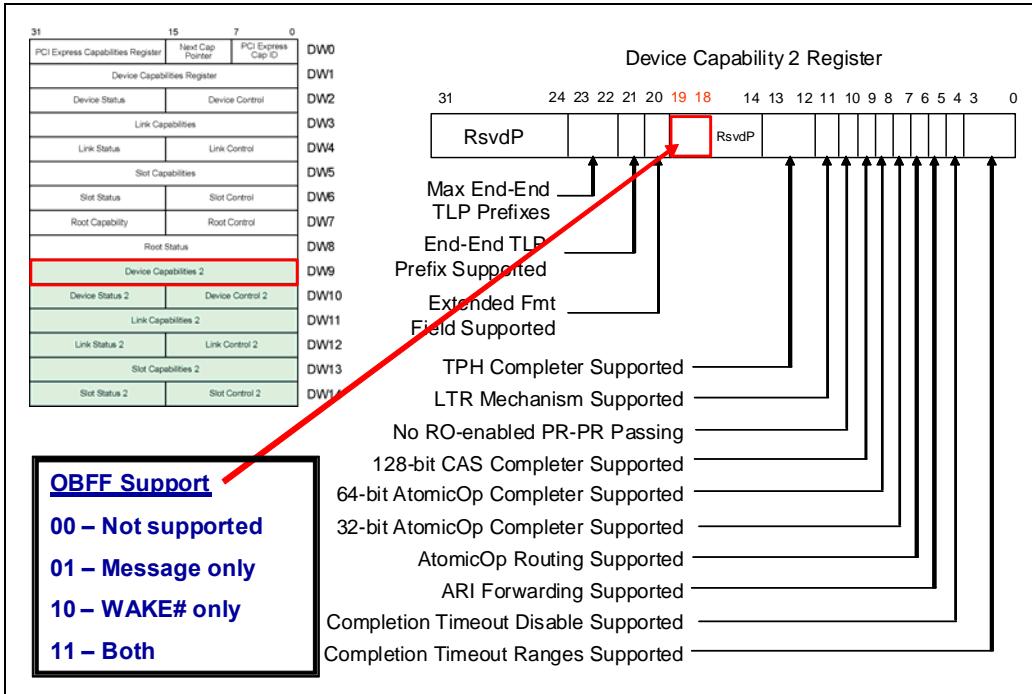
Figure 16-35: OBFF Message Contents



Support for OBFF is indicated via the Device Capability 2 register (Figure 16-36 on page 782), and enabled using the Device Control 2 register (Figure 16-37 on page 783). Note that both the pin and message options may be available. However, the pin method is preferred because it is the lower power option.

Note that there are two variations for enabling a component to forward OBFF messages, and the difference between them has to do with handling a targeted Link that's not in L0. In Variation A, the message will only be sent if the Link is in L0. If it's not, the message is simply dropped to avoid the cost of waking the Link. This is preferred for Downstream Ports when the Device below it is not expected to have time-critical communication requirements and can indicate its need for non-urgent attention by simply returning the Link to L0. For Variation B, the message will always be forwarded and the Link will be returned to L0. This variation is preferred when the downstream Device can benefit from timely notification of the platform state.

Figure 16-36: OBFF Support Indication

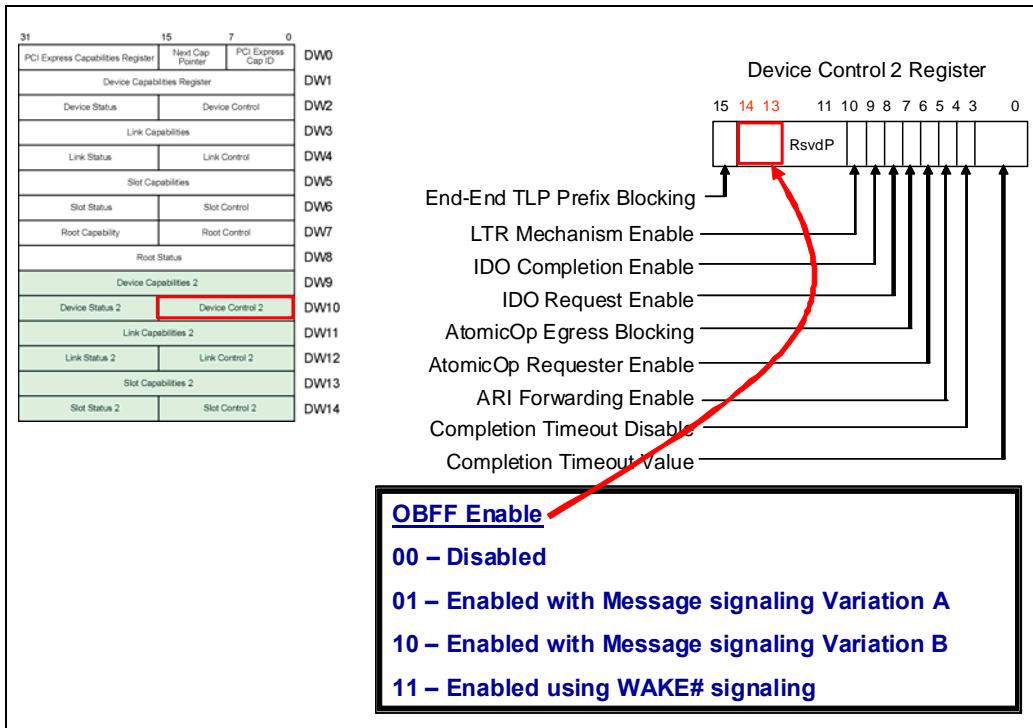


When using WAKE#, enabling any Root Port to assert it is considered a global enable unless there are multiple WAKE# signals, in which case only those associated with that Port are affected. When using the OBFF message, enabling a Root Port only enables the messages on that Port. The expectation in the spec is that all Root Ports would normally be enabled if any of them are, so as to ensure that the whole platform was enabled. However, selectively enabling some Ports and not others is permitted.

When enabling Ports for OBFF, the spec recommends that all Upstream Ports be enabled before Downstream Ports, and Root Ports be enabled last of all. For unpopulated hot plug slots this isn't possible. For that case enabling OBFF using the WAKE# pin to the slot is permitted, but it's recommended that the Downstream Port above the slot not be enabled to deliver OBFF messages.

# Chapter 16: Power Management

Figure 16-37: OBFF Enable Register



Finally, let's refer back to the earlier example in Figure 16-33 on page 778 to consider what these registers might look like for that case. The Downstream Port of the switch that connects to the lower switch will have a value for OBFF Support of 01b - Message Only, while its Upstream Port might have a value of 11b - Both. These values might be hard coded into the device or hardware initialized in some other fashion to make them visible to software after a reset. The Downstream Port would need to have an OBFF Enable value of 01b or 10b - Enabled with Message variation A or B so it could deliver an OBFF message. The Upstream Port would expect to have an OBFF Enable value of 11b - Enabled with WAKE# signaling. The spec points out that when a switch is configured to use the different methods when going from one Port to another, it's required to make the translation and forward the indications.

## LTR (Latency Tolerance Reporting)

The second new feature added to improve PM efficiency is called Latency Tolerance Reporting (LTR). This optional capability allows devices to report the delay they can tolerate when requesting service from the platform so that PM policies for platform resources like main memory can take that into consideration. If software supports it, this provides good performance for devices when they need it and lower power for the system when they don't need a fast response. One simple way of using this information would be to allow the system to postpone waking up to service a request as long as the latency tolerance was still met.

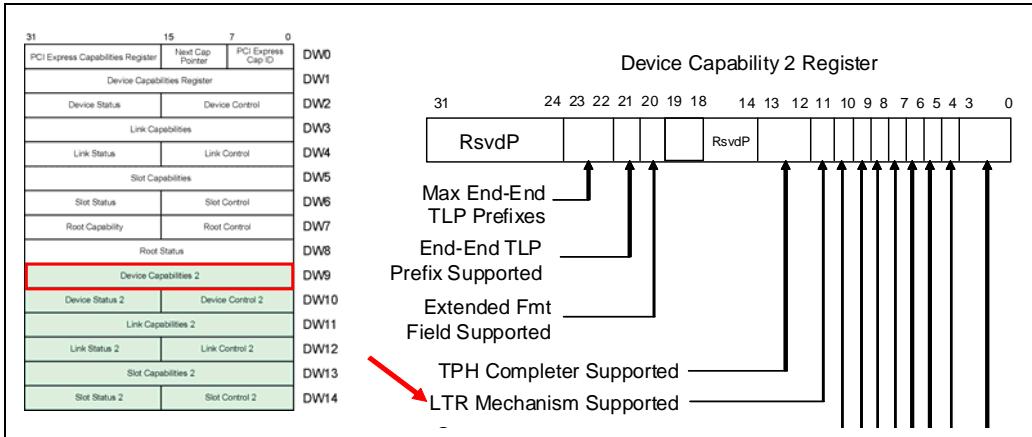
The meaning of “latency tolerance” is not made explicitly clear in the spec, but some things are mentioned that might play into it. For example, the latency tolerance may affect acceptable performance or it may impact whether the component will function properly at all. Clearly, such a distinction would make a big difference in designing a PM policy. Similarly, the device may use buffering or other techniques to compensate for latency sensitivity and knowledge of that would be useful for software.

### LTR Registers

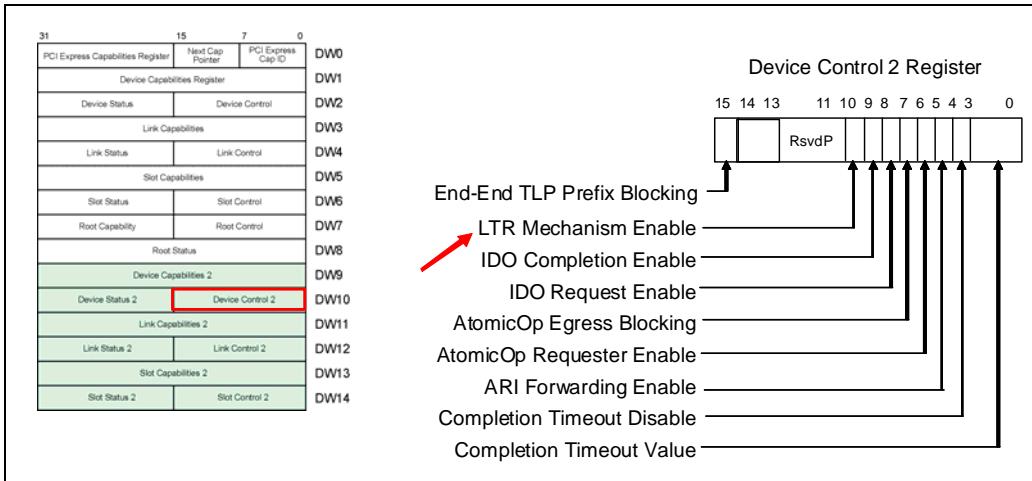
The LTR capability in a device is discovered using a new bit in the PCIe Device Capability 2 Register, as shown in Figure 16-38 on page 785, and enabled in the Device Control 2 Register, illustrated in Figure 16-39 on page 785. The spec prescribes a sequence for enabling LTR, too: devices closest to the Root must be enabled first, working down to the Endpoints. An Endpoint must not be enabled unless its associated Root Port and all intermediate switches also support LTR and have been enabled to service it. It's permissible for some Endpoints to support LTR while others do not. If a Root Port or switch Downstream Port receives an LTR message but doesn't support it or hasn't been enabled yet, the message must be treated as an Unsupported Request. It's recommended that Endpoints send an LTR message shortly after being enabled to do so. It's strongly recommended that Endpoints not send more than two LTR messages within any 500  $\mu$ s period unless required by the spec. However, if they do, Downstream Ports must properly handle them and not generate an error based on that.

## Chapter 16: Power Management

Figure 16-38: LTR Capability Status



*Figure 16-39: LTR Enable*



The target for LTR information is the Root Complex. Participating downstream devices all report their values but the Port just uses the smallest value that was reported as the latency limit for all devices accessed through that Port. The Root is not required to honor requested service latencies but is strongly encouraged to do so.

## LTR Messages

The LTR message itself has the format shown in Figure 16-40 on page 788, where it can be seen that the Routing type 100b (point-to-point) and the LTR message code is 0001 0000b. Two latency values are reported, one for Requests that must be snooped and another for Requests that will not be snooped and therefore should complete more quickly. As seen in the diagram, the format for both is the same and includes the following fields:

- Latency Value and Scale - combine to give a value in the range from 1ns to about 34 seconds. Setting these fields to all zeros indicates that any delay will affect the device and thus the best possible service is requested. The meaning of the latency is defined as follows:
  - For Read Requests, it's the delay from sending the END symbol in the Request TLP until receiving the STP symbol in the first Completion TLP for that Request.
  - For Write Requests, it relates to Flow Control back-pressure. If a write has been issued but the next write can't proceed due to a lack of Flow Control credits, the latency is the time from the last symbol of that write (END) until the first symbol of the DLLP that gives more credits (SDP). In other words, this represents the time within which the Root Port should be able to accept the next write.
- Requirement - can be set for none, or one, or both to indicate whether that latency value is required. If a device doesn't implement one of these traffic types or has no service requirements for it, then this bit must be cleared for the associated field. If a device has reported requirements but has since been directed into a device power state lower than D0, or if its LTR Enable bit has been cleared, the device must send another LTR message reporting that these latencies are no longer required.

## Guidelines Regarding LTR Use

Endpoints have a few guidelines regarding the use of LTR:

1. It's recommended that they send an updated LTR message every time their service requirements change, and the spec spends some time going over examples of this. The bottom line here is that devices need to take all the delays into account when making a change to the service requirements. That accounting includes time for the reference clock to be restored if was turned off, for the Link to be brought back to L0, for the LTR message to be delivered, and for the platform to prepare to handle the new requirement.
2. If the latency tolerance is being reduced, it's recommended that the LTR message be sent far enough ahead of the first associated Request to ensure that the platform is ready.

## **Chapter 16: Power Management**

---

3. If the latency tolerance is being increased, then the LTR message to report that should immediately follow the final Request that used the previous latency value.
4. To achieve the best overall platform power efficiency, it's recommended that Endpoints buffer Requests as much as they can and then send them in bursts that are as long as the Endpoint can support.

Multi-Function Devices (MFDs) have a few rules of their own. For example, they must send a "conglomerated" LTR message as follows:

1. Reported latency values must reflect the lowest values associated with any Function. The snoop and no-snoop latencies could be associated with different Functions, but if none of them have a requirement for snoop or no-snoop traffic, then the requirement bit for that type must not be set.
2. MFDs must send a new LTR message upstream if any of the Functions changes its values in a way that affects the conglomerated value.

Switches have a similar set of rules related to LTR. Basically, they collect the messages from Downstream Ports that have been enabled to use LTR and send a "conglomerated" message upstream according to the following rules:

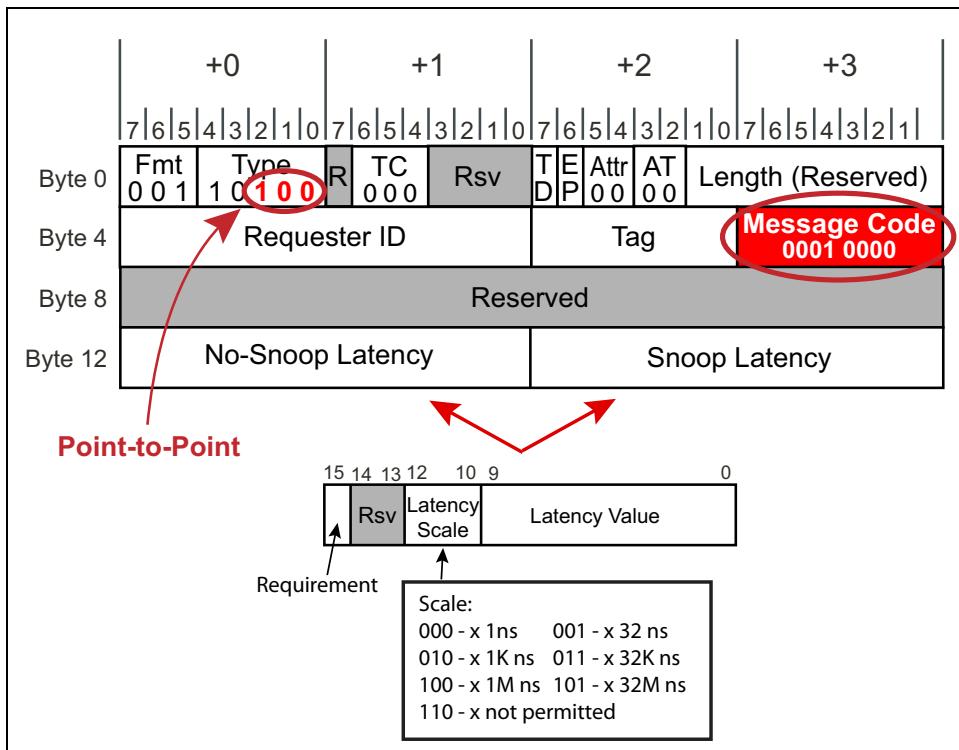
1. If the Switch supports LTR, it must support it on all of its Ports.
2. The Upstream Port is allowed to send LTR messages only when the LTR Enable bit is set or shortly after software has cleared it so it can report that any previous requirements are no longer in effect.
3. The conglomerated LTR value is based on the lowest value reported by any participating Downstream Port. If the Requirement bit is clear, or an invalid value is reported, the latency is considered effectively infinite.
4. If any Downstream Port reports that an LTR value is required, the Requirement bit will be set for that type in the LTR message forwarded upstream.
5. The LTR values reported upstream must take into account the latency of the Switch itself. If the Switch latency changes based on its operational mode, it must not be allowed to exceed 20% of the minimum value reported on all Downstream Ports. The value reported on the Upstream Port is the minimum reported value on all the Downstream Ports minus the Switch's own latency, although the value can't be less than zero.
6. If a Downstream Port goes to DL\_Down status, previous latencies for that Port must be treated as invalid. If that changes the conglomerated values upstream then a new message must be sent to report that.
7. If a Downstream Port's LTR Enable bit is cleared, any latencies associated with that Port must be considered invalid, which may also result in a new LTR message being sent upstream.
8. If any Downstream Ports receive new LTR values that would change the conglomerated value, the Switch must send a new LTR message upstream to report that.

# PCI Express Technology

Finally, the Root Complex also has a few rules related to LTR:

1. The RC is allowed to delay processing of a device Request as long as it satisfies the service requirements. One application of this might be to buffer up several Requests from an Endpoint and service them all in a batch.
2. If the latency requirements are updated while a series of Requests is in progress, the new values must be comprehended by the RC prior to servicing the next Request, and within less time than the previously reported latency requirements.

Figure 16-40: LTR Message Format

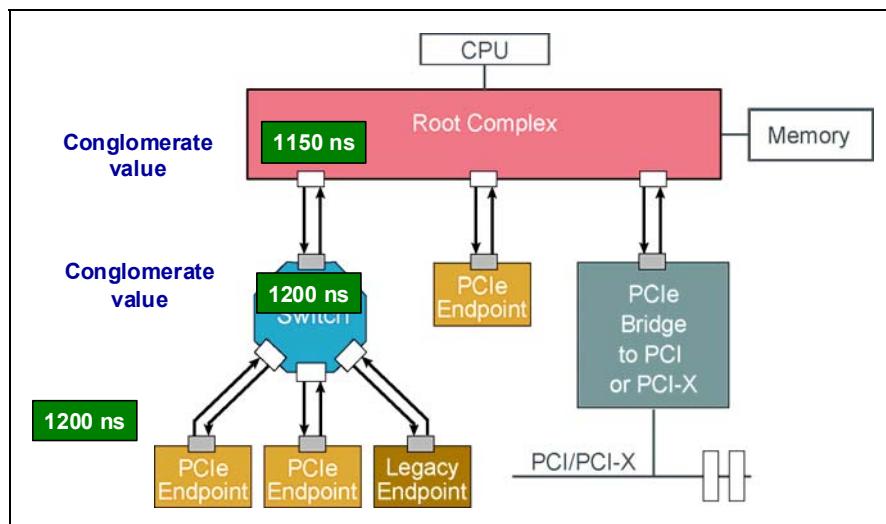


# Chapter 16: Power Management

## LTR Example

To illustrate the concepts discussed so far, consider the example topology shown in Figure 16-41 on page 789. Here, the Endpoint on the lower left has delivered an LTR message to the Switch reporting a Snoop Latency requirement of 1200ns. At this point, none of the other Endpoints connected to the Switch has reported an LTR value, so that becomes the conglomerated value to be reported upstream. However, the Switch has an internal latency of 50ns so that must be subtracted from the value to be reported, resulting in the Upstream Port sending an LTR message reporting 1150ns to the Root Port.

Figure 16-41: LTR Example

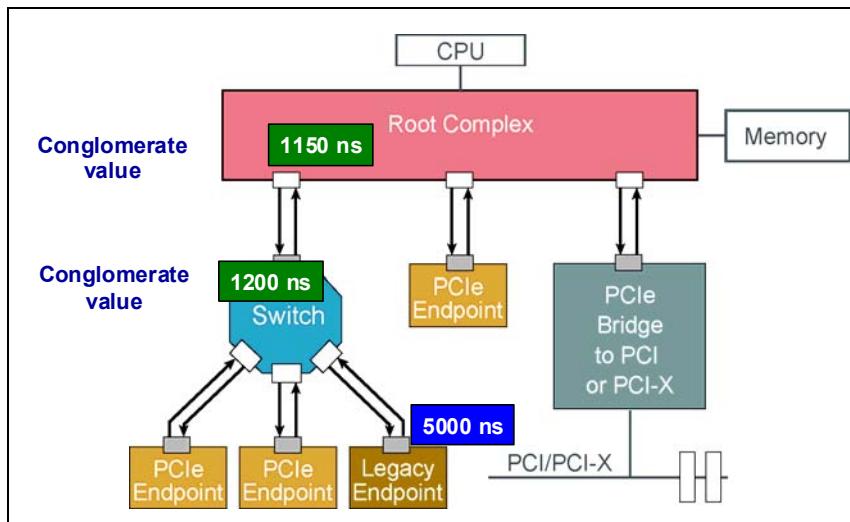


Next, the Legacy Endpoint delivers an LTR message with a large latency requirement of 5000ns, as shown in Figure 16-42 on page 790. Since this is larger than the current conglomerate value for the Switch, no LTR message is sent for this case.

# PCI Express Technology

---

Figure 16-42: LTR - Change but no Update



In the next stage, the middle Endpoint reports its LTR value as 700ns. This is smaller than the current conglomerate value, so the Switch calculates the new value of 650ns by subtracting its internal latency and forwards that upstream as an LTR message. That makes the current latency requirement for that Root Port 650ns, as seen in Figure 16-43 on page 791.

Finally, the Link to the middle Endpoint stops working for some reason as shown in Figure 16-44 on page 791, and the Switch Port reports DL\_Down. Consequently, the LTR value for that Port must be considered invalid. Since its value was being used as the current conglomerate value, the conglomerate will be updated to the lowest value that is still valid, which is the 1200ns reported by the left-most Endpoint. The Switch will then subtract its internal latency and report 1150ns to the Root Port with a new LTR message.

# Chapter 16: Power Management

Figure 16-43: LTR - Change with Update

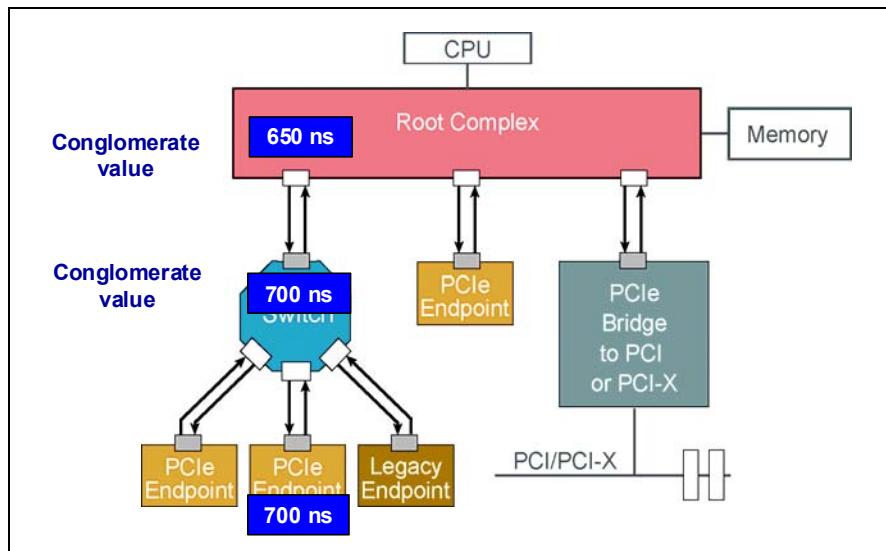
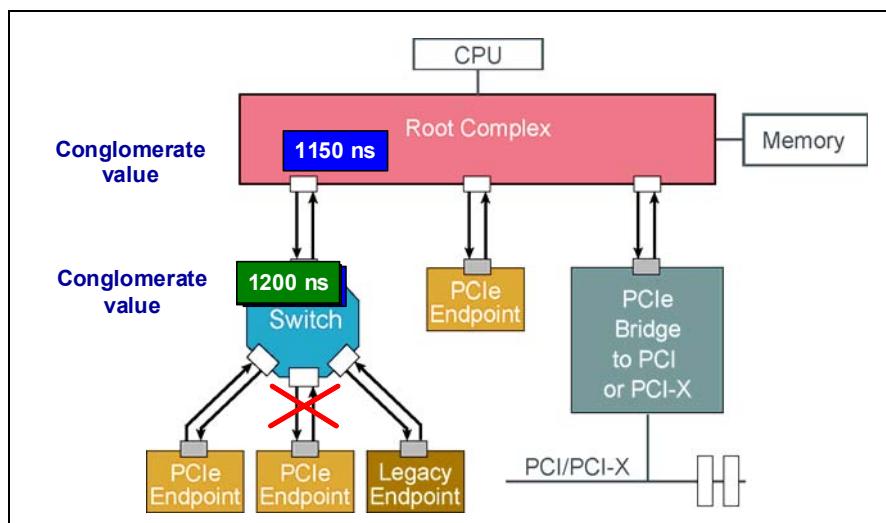


Figure 16-44: LTR - Link Down Case



## **PCI Express Technology**

---

---

---

# 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](http://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 (eXtended).

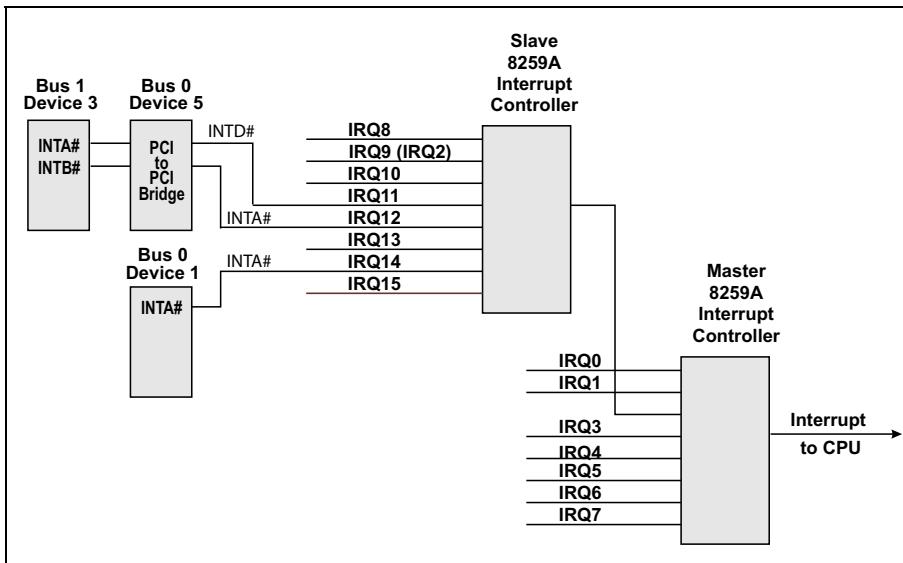
**Legacy PCI Interrupt Delivery** — This original mechanism defined for the PCI bus consists of up to four signals per device or INTx# (INTA#, INTB#, INTC#, and INTD#) as shown in Figure 17-1 on page 795. In this model, the pins are shared by wire-ORing them together, and they'd eventually be connected 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

# Chapter 17: Interrupt Support

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

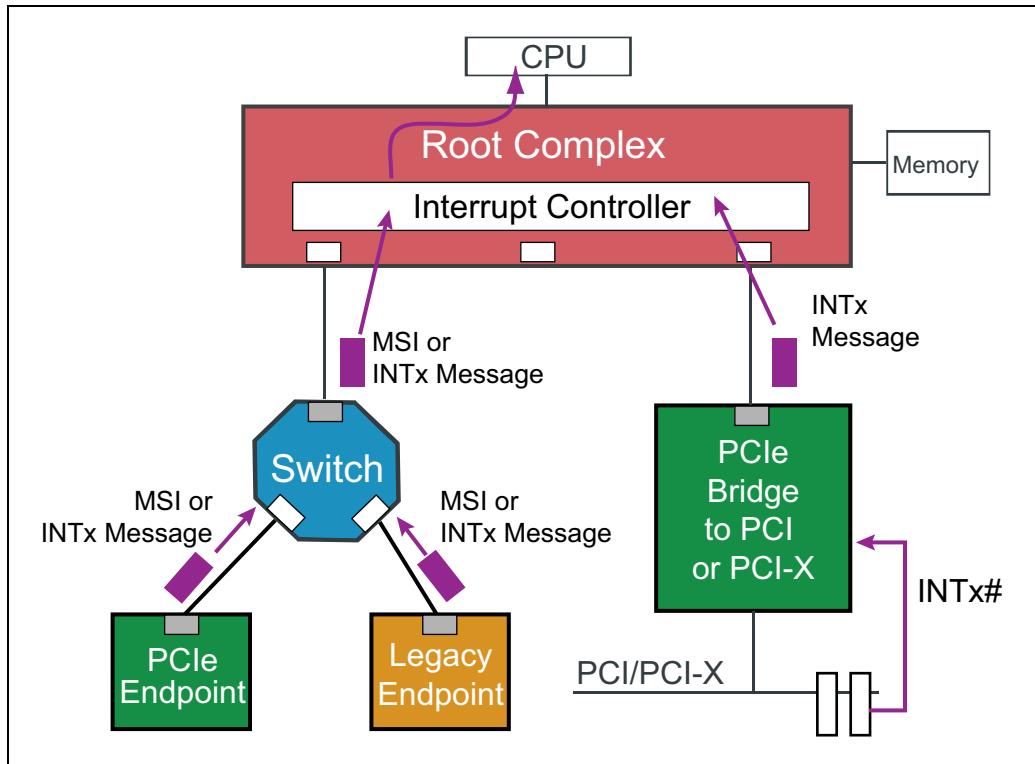


**MSI Interrupt Delivery** — MSI eliminates the need for sideband signals by using memory writes to deliver the interrupt notification. The term “Message Signaled Interrupt” can be confusing because its name includes the term “Message” which is a type of TLP in PCIe, but an MSI interrupt is a Posted 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 FEE<sub>x</sub>\_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.

# PCI Express 3.0 Technology

Figure 17-2: Interrupt Delivery Options in PCIe System



## The Legacy Model

### General

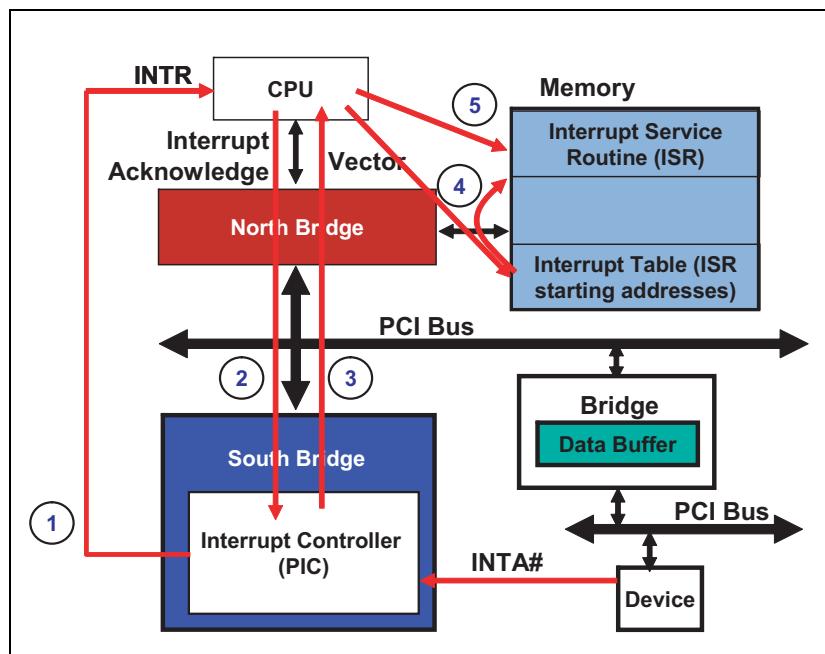
To illustrate the legacy interrupt delivery model, refer to Figure 17-3 on page 797 and consider the usual steps involved in interrupt delivery using the legacy method of interrupt pins:

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.

# Chapter 17: Interrupt Support

2. Once the CPU detects the assertion of INTR and is ready to act on it, it must identify which interrupt actually needs service, and that is done by the CPU issuing a special command on the processor bus called an Interrupt Acknowledge.
3. This command is routed by the system to the PIC, which returns an 8-bit value called the Interrupt Vector to report the highest priority interrupt currently pending. A unique vector would have been programmed earlier by system software for each IRQ input.
4. The interrupt handler then uses the vector as an offset into the Interrupt Table (an area set up by software to contain the start addresses of all the Interrupt Service Routines, ISRs), and fetches the ISR start address it finds at that location.
5. That address would point to the first instruction of the ISR that had been set up to handle this interrupt. This handler would be executed, servicing the interrupt and telling its device to deassert its INTx# line and then would return control to the previously interrupted task.

Figure 17-3: Legacy Interrupt Example



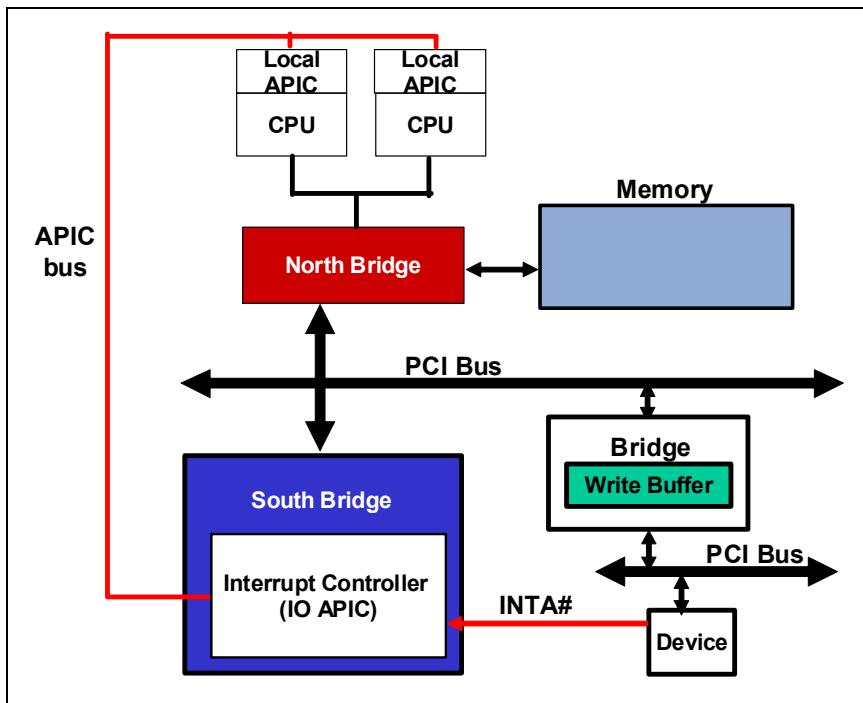
## Changes to Support Multiple Processors

This model works well for single-CPU systems, but has a limitation that makes it sub-optimal in a multi-CPU system. The problem is that the INTR pin can only be connected to one CPU. If multiple processors are present then only one of them will see the interrupts and will have to service them all while the other CPUs won't see any of them. To obtain the best performance, such systems really need an even distribution of the system tasks across all the processors, referred to as SMP (Symmetric Multi-Processing) but the pin model won't support it.

To achieve better SMP, a new model was needed, and toward this end the PIC was modified to become the IO APIC (Advanced Programmable Interrupt Controller). The IO APIC was designed to have a separate small bus, called the APIC Bus, over which it could deliver interrupt messages, as shown in Figure 17-4 on page 799. In this model, the message contained the interrupt vector number, so there was no need for the CPU to send an Interrupt Acknowledge down into the IO world to fetch it. The APIC Bus connected to a new internal logic block within the processors called the Local APIC. The bus was shared among all the agents and any of them could initiate messages on it but, for our purposes, the interesting part is its use for interrupt delivery from peripherals. Those interrupts could now be statically assigned by software to be serviced by different CPUs, multiple CPUs or even dynamically assigned by the IO APIC.

# Chapter 17: Interrupt Support

Figure 17-4: APIC Model for Interrupt Delivery



That model, known as the APIC model, was sufficient for several years but still depended on sideband pins from the peripheral devices to work. Another limitation of this model was the number of IRQs (interrupt request lines) into the IO APIC. Without a very large number of IRQs, peripheral devices had to share IRQs which means added latency anytime that IRQ is asserted because there could be multiple devices that could have asserted it and software must evaluate all of them. This technique of linking multiple ISRs together was often referred to as interrupt chaining. Eventually, because of this issue and a couple other minor issues, another improvement came along.

Why not have the peripheral devices themselves send interrupt messages directly to the Local APICs? All that is needed is a communications path which already exists in the form of the PCI bus and the processor bus. So the APIC bus was eliminated and all interrupts were delivered to the Local APICs in the form of memory writes, referred to as MSIs or Message Signaled Interrupts. These MSIs were targeting a special address that the system understood to be an interrupt message targeting the Local APICs. (This special address address was tra-

# PCI Express 3.0 Technology

---

ditionally FEE<sub>x</sub>\_xxxxh for x86-based systems.) Even the IO APIC was programmed to send its interrupt notifications over the ordinary data bus using memory writes (MSI). Now it simply sends an MSI memory write across the data bus targeting the memory address of the desired processor's Local APIC, and that has the effect of notifying the processor of the interrupt.

This model is known as the xAPIC model, and since it is not based on sideband signals which go into an interrupt controller with a limited number of inputs, the need to share interrupts is almost eliminated. More information can be found about this model in "An MSI Solution" on page 827.

PCI added MSI support as an option years ago and PCIe made that capability a requirement. A peripheral that can generate MSI transactions on its own opens new options for handling interrupts, such as giving each Function the ability to generate multiple unique interrupts instead of just one.

---

## Legacy PCI Interrupt Delivery

This section provides more detail on legacy PCI interrupt delivery. Readers familiar with PCI may wish to proceed to "Virtual INTx Signaling" on page 805 to learn more about how PCIe emulates this legacy model, or to "The MSI Model" on page 812 to learn more about that method.

PCI devices that use interrupts have two options. They may use either:

- INTx# active low-level signals that can be shared and were defined in the original spec.
- Message Signaled Interrupts that were added as an option with the 2.2 version of the spec. MSI needs no modification for use in a PCIe system.

### Device INTx# Pins

A PCI device can implement up to 4 INTx# signals (INTA#, INTB#, INTC#, and INTD#). More than one pin is available because PCI devices can support up to 8 functions, each of which is allowed to drive one (but only one) interrupt pin. When PCI was developed, a typical system used a chipset that included the 15-input 8259 PIC, so that's how many IRQs (which map to interrupt vectors) that were available to the system. However, many of those were already used for system purposes like the system timer, keyboard interrupt, mouse interrupt, and so on. In addition, some pins were reserved for ISA cards that could still be plugged into these older systems. Consequently, the PCI spec writers considered that only four IRQs would reliably be available for their new bus, and so the spec only supported four interrupt pins. However, as you probably know, there are typically more than four PCI devices on a PCI bus and even a single device could have more than four functions inside, each wanting its own inter-

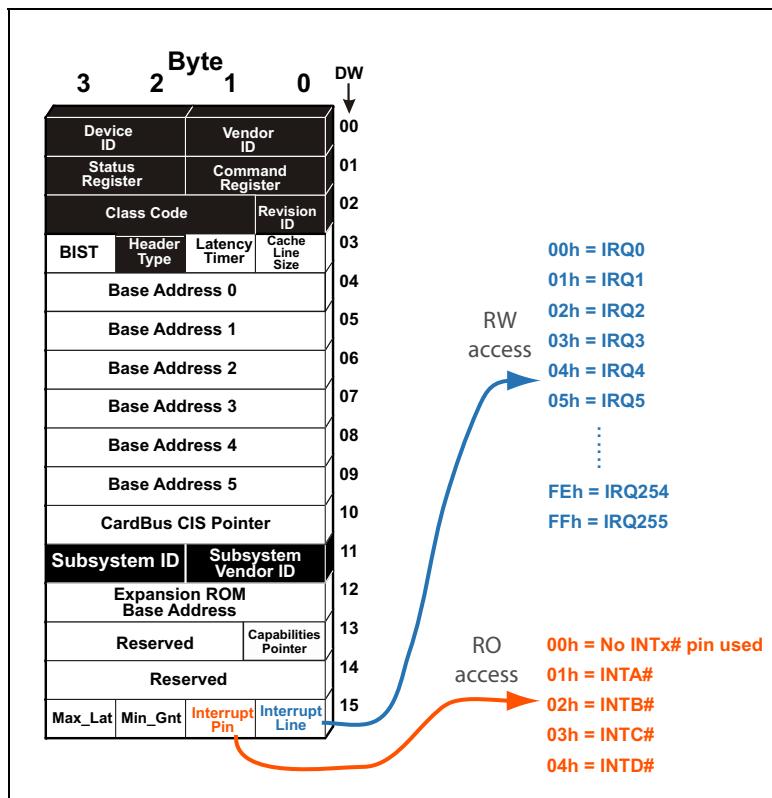
# Chapter 17: Interrupt Support

rupt. These reasons are why the PCI interrupts were designed to be level-sensitive and shareable. These signals could simply be wire-ORed together to get down to a handful of resulting outputs, each one representing interrupt requests. Since they are shared, when an interrupt is detected, the interrupt handler software will need to go through the list of functions that are sharing the same pin and test to see which ones need servicing.

## Determining INTx# Pin Support

PCI functions indicate support for an INTx# signal in their configuration headers. The read-only Interrupt Pin register illustrated in Figure 17-5 indicates whether an INTx# is supported by this function and if so, which interrupt pin will it assert when requesting an interrupt.

Figure 17-5: Interrupt Registers in PCI Configuration Header



## Interrupt Routing

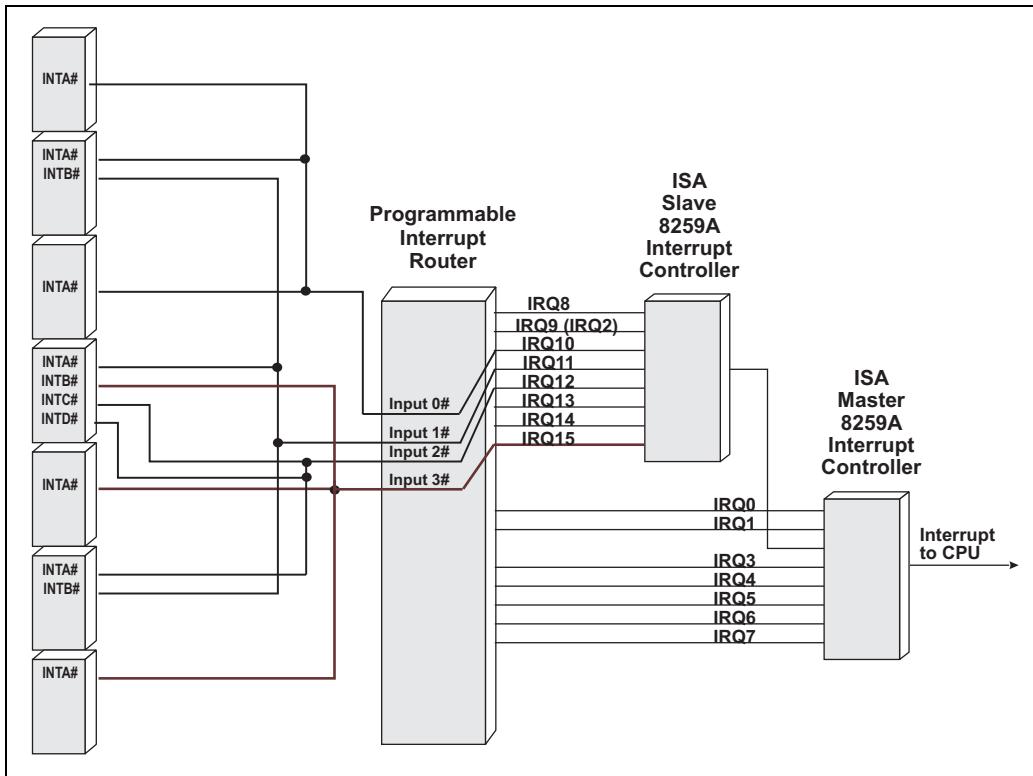
The Interrupt Line register shown in Figure 17-5 on page 801 gives the next information that a driver needs to know: the input pin of the PIC to which this pin has been connected. The PIC is programmed by system software with a unique vector number for each input pin (IRQ). The vector for the highest-priority interrupt asserted is reported to the processor who then uses that vector to index into a corresponding entry in the interrupt vector table. This entry points to the interrupting device's interrupt service routine which the processor executes.

The platform designer assigns the routing of INTx# pins from devices. They can be routed in a variety of ways, but ultimately each INTx# pin connects to an input of the interrupt controller. Figure 17-6 on page 803 illustrates an example in which several PCI device interrupts are connected to the interrupt controller through a programmable router. All signals connected to a given input of the programmable router will be directed to a specific input of the interrupt controller. Functions whose interrupts are routed to a common interrupt controller input will all have the same Interrupt Line number assigned to them by platform software (typically firmware). In this example, IRQ15 has three PCI INTx# inputs from different devices connected to it. Consequently, the functions using these INTx# lines will share IRQ15 and will therefore all cause the controller to send the same vector when queried. That vector will have the three ISRs for the different Functions chained together.

## Associating the INTx# Line to an IRQ Number

Based on system requirements, the router is programmed to connect its four inputs to four available PIC inputs. Once this is done, the routing of the INTx# pin associated with each function is known and the Interrupt Line number is written by software into each Function. The value is ultimately read by the Function's device driver so it will know which interrupt table entry it has been assigned. That's the place where the starting address of its ISR will be written, a process referred to as "hooking the interrupt". When this function later generates an interrupt, the CPU will receive the vector number that corresponds to the IRQ specified in the Interrupt Line register. The CPU uses this vector to index into the interrupt vector table to fetch the entry point of the interrupt service routine associated with the Function's device driver.

Figure 17-6: INTx Signal Routing is Platform Specific



## INTx# Signaling

The INTx# lines are active-low signals implemented as open-drain with a pull-up resistor provided on each line by the system. Multiple devices connected to the same PCI interrupt request signal line can assert it simultaneously without damage.

When a Function signals an interrupt it also sets the Interrupt Status bit located in the Status register of the config header. This bit can be read by system software to see if an interrupt is currently pending. (See Figure 17-8 on page 804.)

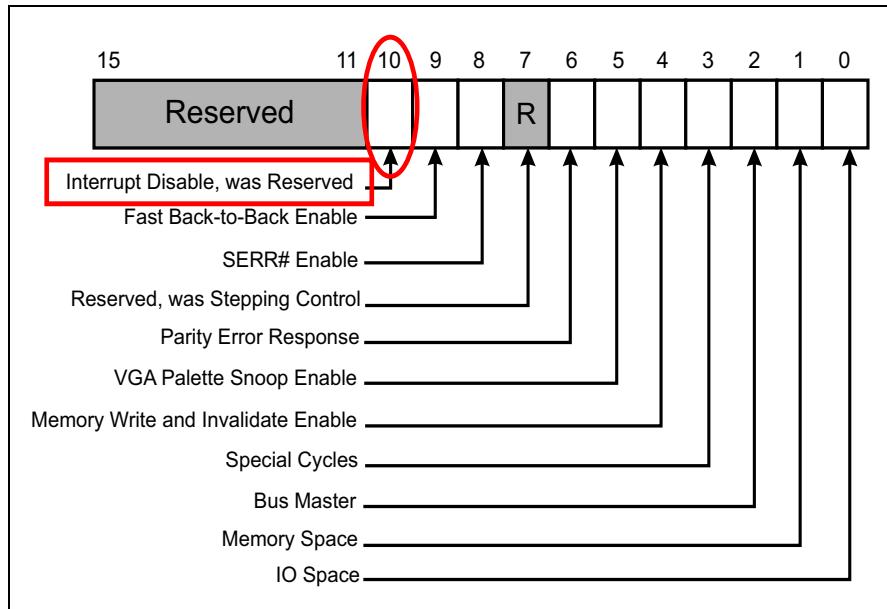
**Interrupt Disable.** The 2.3 PCI spec added an Interrupt Disable bit (Bit 10) to the Command register of the config header. See Figure 17-7 on page 804. The bit is cleared at reset permitting INTx# signal generation, but software may set it

# PCI Express 3.0 Technology

to prevent that. Note that the Interrupt Disable bit has no effect on Message Signalled Interrupts (MSI). MSIs are enabled via the Command Register in the MSI Capability structure. Enabling MSI automatically has the effect of disabling interrupt pins or emulation.

**Interrupt Status.** The PCI 2.3 spec added a read-only Interrupt Status bit to the configuration status register (pictured in Figure 17-8 on page 805). A function must set this status bit when an interrupt is pending. In addition, if the Interrupt Disable bit in the Command register of the header is cleared (i.e. interrupts enabled), then the function's INTx# signal is asserted when this status bit is set. This bit is unaffected by the state of the Interrupt Disable bit.

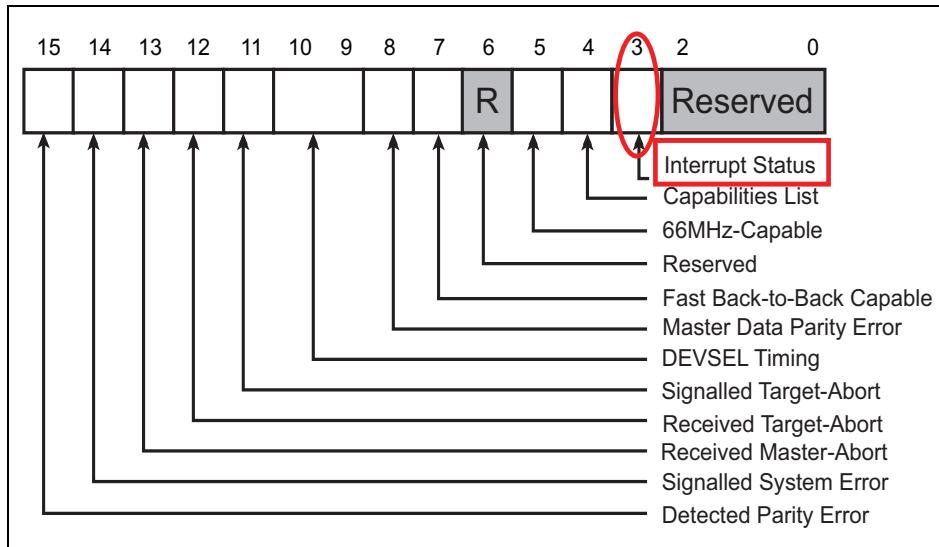
Figure 17-7: Configuration Command Register — Interrupt Disable Field



# Chapter 17: Interrupt Support

---

Figure 17-8: Configuration Status Register — Interrupt Status Field



---

## Virtual INTx Signaling

### General

If circumstances make the use of MSI not possible in a PCIe topology, the INTx signaling model would be used. Following are two examples of devices that would need to be able to use INTx messages:

**PCIe-to-(PCI or PCI-X) bridges** — Most PCI devices will use the INTx# pins because MSI support is optional for them. Since PCIe doesn't support sideband interrupt signaling, the inband messages are used instead. The interrupt controller understands the message and delivers an interrupt request to the CPU which would include a pre-programmed vector number.

**Boot Devices** — PC systems commonly use the legacy interrupt model during the boot sequence because MSI usually requires OS-level initialization. Generally, a minimum of three subsystems are needed for booting: an output to the operator such as video, an input from the operator which is typically the keyboard, and a device that can be used to fetch the OS, typically a hard drive. PCIe devices involved in initializing the system are called "boot devices." Boot devices will use legacy interrupt support until the OS and device drivers are loaded, after which it's preferable they use MSI.

# PCI Express 3.0 Technology

---

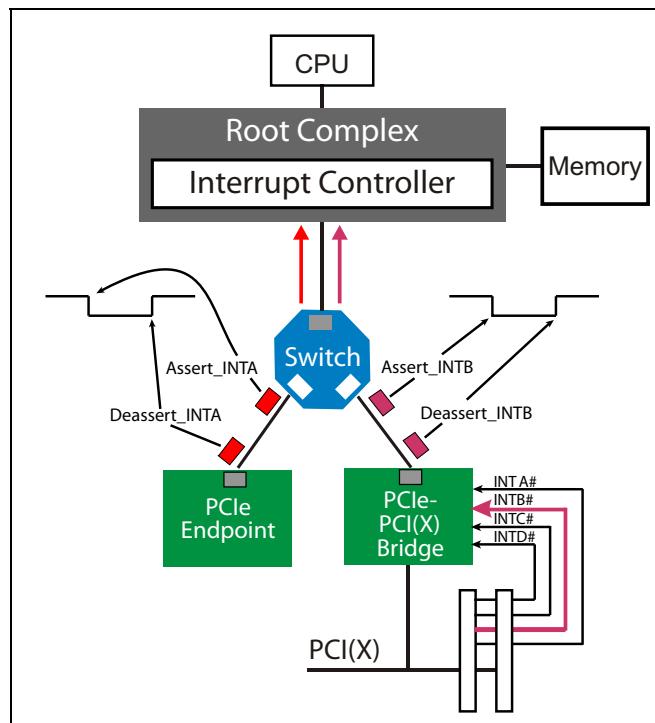
## Virtual INTx Wire Delivery

Figure 17-9 on page 806 illustrates a system with a PCIe Endpoint and a PCIe-to-PCI Bridge. If we assume software has not enabled MSI on the Endpoint, it will deliver interrupt requests with INTx messages. In this example, the bridge is propagating pin-based interrupts from connected PCI devices with INTx messages. As can be seen, the bridge sends an INTB messages to signal the assertion and deassertion of its INTB# input from the PCI bus. The PCIe Endpoint is shown signaling an INTA using emulation messages. Note that INTx# signaling involves two messages:

- **Assert\_INTx** messages indicate a high-to-low transition (from inactive to active) of the virtual INTx# signal.
- **Deassert\_INTx** messages indicate a low-to-high transition.

When a Function delivers an Assert\_INTx message, it also sets its Interrupt Status bit in the Configuration Status register, just as it would if it asserted the physical INTx# pin (see Figure 17-8 on page 805).

*Figure 17-9: Example of INTx Messages to Virtualize INTA#-INTD# Signal Transitions*



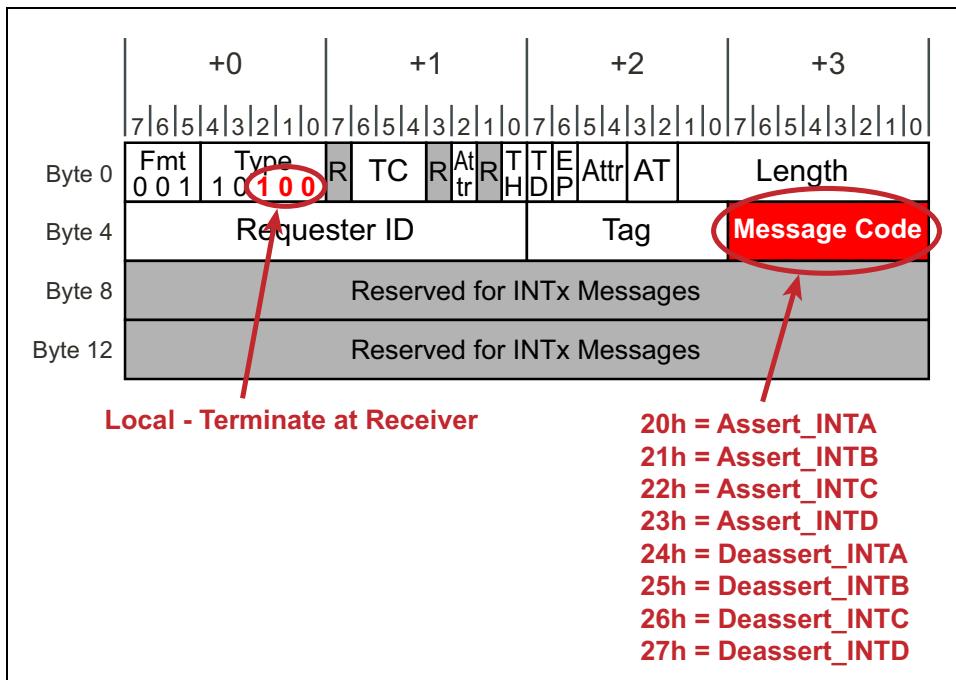
# Chapter 17: Interrupt Support

## INTx Message Format

Figure 17-10 on page 807 depicts the format of the INTx message header. The interrupt controller is the ultimate destination of these messages, however the routing method employed is *not* “Route to the Root Complex”, but is actually “Local - Terminate at Receiver” as shown in Figure 17-10. There are two reasons for this. The first is because each bridge (including Switch Ports and Root Ports) along the upstream path may map the virtual interrupt wire to a different virtual interrupt wire across the bridge (e.g., a Switch Port receives Assert\_INTA but maps it to Assert\_INTB when propagating it upstream). More info about this INTx mapping can be found in “INTx Mapping” on page 808.

The second reason for the local routing type of these messages is due to the fact that we’re emulating a pin-based signal. If a port receives an assert interrupt message that maps to INTA on its primary side and it has already sent an Assert\_INTA message upstream because of a previous interrupt, then there is no reason to send another one. INTA is already seen as asserted. More info about this collapsing of INTx messages can be found in “INTx Collapsing” on page 810.

Figure 17-10: INTx Message Format and Type



## Mapping and Collapsing INTx Messages

### INTx Mapping

Switches must adhere to the INTx mapping defined by the PCI spec, shown in Table 17-1 on page 809. This mapping defines the virtual connection that exists when interrupts are routed across a PCI-to-PCI bridge. The mapping is based on the INTx message type and the Device number from the Requester ID field in the message.

Refer to Figure 17-11 on page 810 for this example. The assert interrupt messages received on the two downstream switch ports are both INTA messages. The virtual PCI-to-PCI bridge at each of the ingress ports will map both INTA messages to INTA, meaning no change. This is because the Device number of both originating Endpoint devices is zero (which is contained in the interrupt message itself as part of the Requester ID, ReqID). Table 17-1 shows that interrupts messages coming from Device 0 map to the same INTx message on the other side of the bridge (i.e., internal to the Switch both INTA messages are mapped to INTA). So each downstream port will propagate the interrupt messages upstream without changing their virtual wire. However, the propagated interrupt messages no longer have the ReqID of the original requester, they now have the ReqID of the port that is propagating the interrupt message.

Next, the upstream Switch Port receives the propagated interrupt messages. The INTA interrupt from port 2:1:0 is going to be mapped to an INTB message when propagated upstream because the interrupt message indicates it came from Device 1 (ReqID 2:1:0). The other interrupt being propagated by port 2:2:0 is going to be mapped to an INTC message when sent from the upstream Switch Port to the Root Port. Refer to Table 17-1 to confirm these mappings.

The reason for this interrupt mapping is the same as it was for PCI: to avoid as much as possible having multiple functions sharing the same INTx# pin. As stated previously, single function devices are required to use INTA if using legacy interrupts. So if all the Functions downstream of a Root Port used INTA and there was no mapping across bridges, they would all be routed to the same IRQ. Which means anytime one of the Functions asserted INTA, all the Functions would have to be checked. This would result in significant interrupt servicing latencies for the Functions at the end of the list. This interrupt mapping method is a crude attempt at distributing interrupts (especially INTA) across all four INTx virtual wires because each INTx virtual wire can be mapped to a separate IRQ at the interrupt controller.

## Chapter 17: Interrupt Support

---

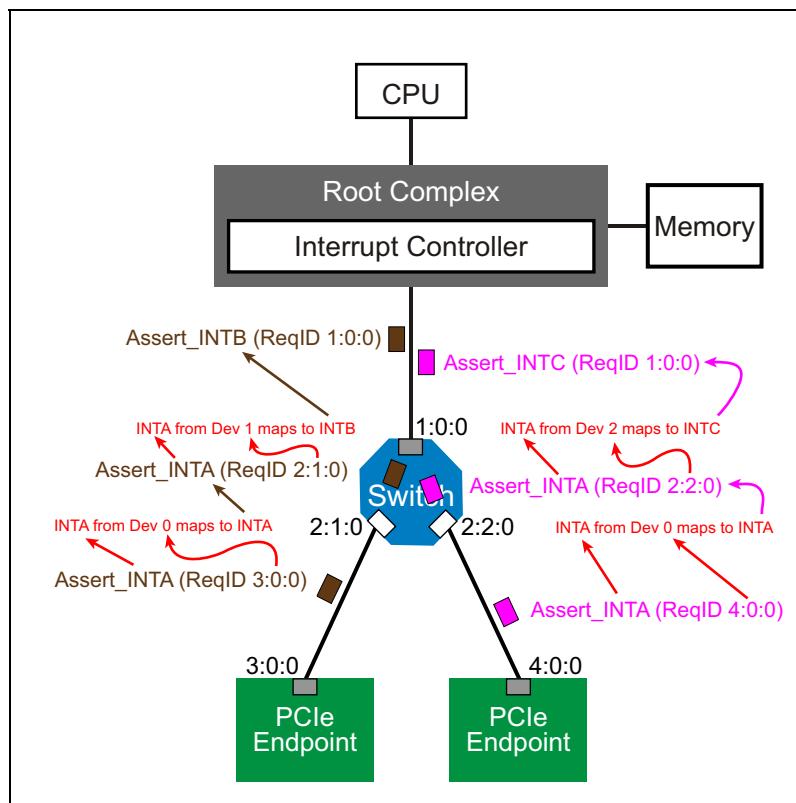
Table 17-1: INTx Message Mapping Across Virtual PCI-to-PCI Bridges

Device Number of Delivering INTx	INTx Message Type at Input	INTx Message Type at Output
0, 4, 8, 12 etc.	INTA	INTA
	INTB	INTB
	INTC	INTC
	INTD	INTD
1, 5, 9, 13 etc.	INTA	INTB
	INTB	INTC
	INTC	INTD
	INTD	INTA
2, 6, 10, 14 etc.	INTA	INTC
	INTB	INTD
	INTC	INTA
	INTD	INTB
3, 7, 11, 15 etc.	INTA	INTD
	INTB	INTA
	INTC	INTB
	INTD	INTC

# PCI Express 3.0 Technology

---

Figure 17-11: Example of INTx Mapping



## INTx Collapsing

PCIe Switches must ensure that INTx messages are delivered upstream in the correct fashion. Specifically, interrupt routing of legacy PCI implementations must be handled such that software can determine which interrupts are routed to which interrupt controller inputs. INTx# lines may be wire-ORed and be routed to the same IRQ input on the interrupt controller, and when multiple devices signal interrupts on the same line, only the first assertion is seen by the interrupt controller. Similarly, when one of these devices deasserts its INTx# line, the line remains asserted until the last one is turned off. These same principles apply to PCIe INTx messages.

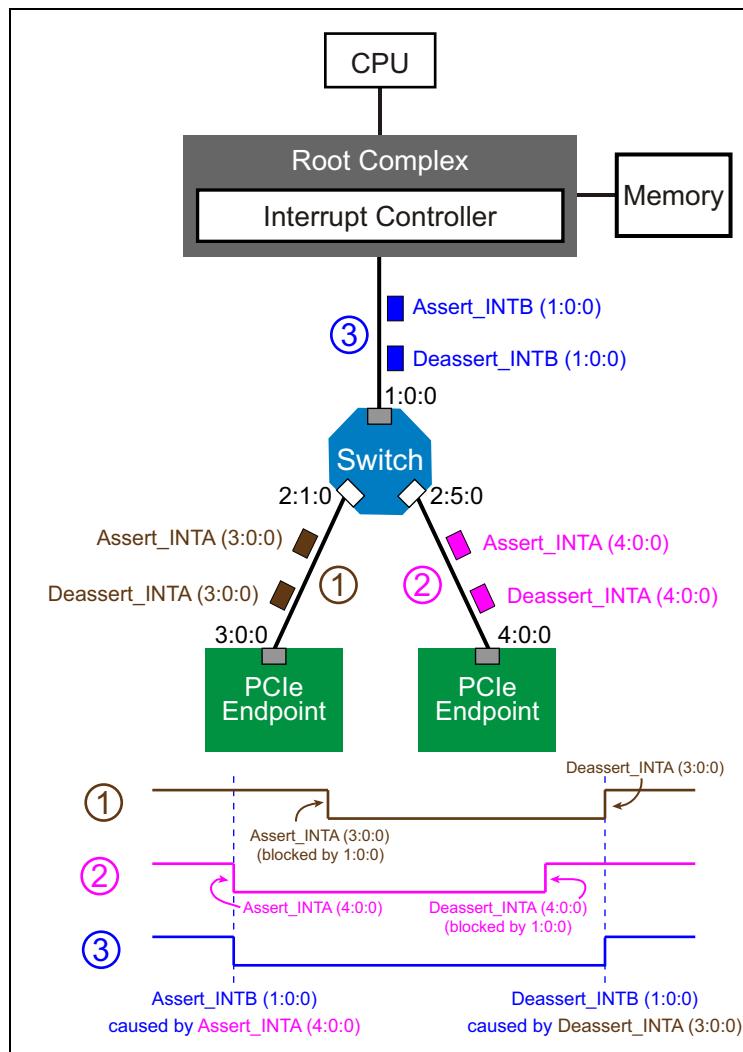
In some cases, however, two overlapping INTx messages may be mapped to the same INTx message by a virtual PCI bridge at the egress port, requiring the messages to be collapsed. Consider the following example illustrated in Figure 17-12 on page 811.

# Chapter 17: Interrupt Support

When the upstream Switch Port maps the interrupt messages for delivery on the upstream link, both interrupts will be mapped as INTB (based on the device numbers of the downstream Switch Ports). Note that because these two overlapping messages are the same they must be collapsed.

Collapsing ensures that the interrupt controller will never receive two consecutive Assert\_INTx or Deassert\_INTx messages for the shared interrupts. This is equivalent to INTx signals being wire-ORed.

Figure 17-12: Switch Uses Bridge Mapping of INTx Messages



## INTx Delivery Rules

The rules associated with the delivery of INTx messages have some unique characteristics:

- Assert\_INTx and Deassert\_INTx are only issued in the upstream direction.
- Switches that are collapsing interrupts will only issue INTx messages upstream when there is a change of the interrupt status.
- Devices on either side of a link must track the current state of INTA-INTD assertion.
- A Switch tracks the state of the four virtual wires for each of its downstream ports, and may present a collapsed set of virtual wires on its upstream port.
- The Root Complex must track the state of the four virtual wires (A-D) for each downstream port.
- INTx signaling may be disabled with the Interrupt Disable bit in the Command Register.
- If any INTx virtual wires are active and device interrupts are then disabled, a corresponding Deassert\_INTx message must be sent.
- If a downstream Switch Port goes to DL\_Down status, any active INTx virtual wires must be deasserted, and the upstream port updated accordingly (Deassert\_INTx message required if that INTx was in active state).

---

## The MSI Model

A PCIe Function indicates MSI support via the MSI Capability registers. Each Function must implement either the MSI Capability Structure or the MSI-X (eXtended MSI, see “The MSI-X Model” on page 821) Capability Structure, or both. The MSI Capability registers are set up by configuration software and include:

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

See “Memory Request Header Fields” on page 188 for a review of the Memory Write Transaction Header. Note that MSIs always have a data payload of 1DW.

---

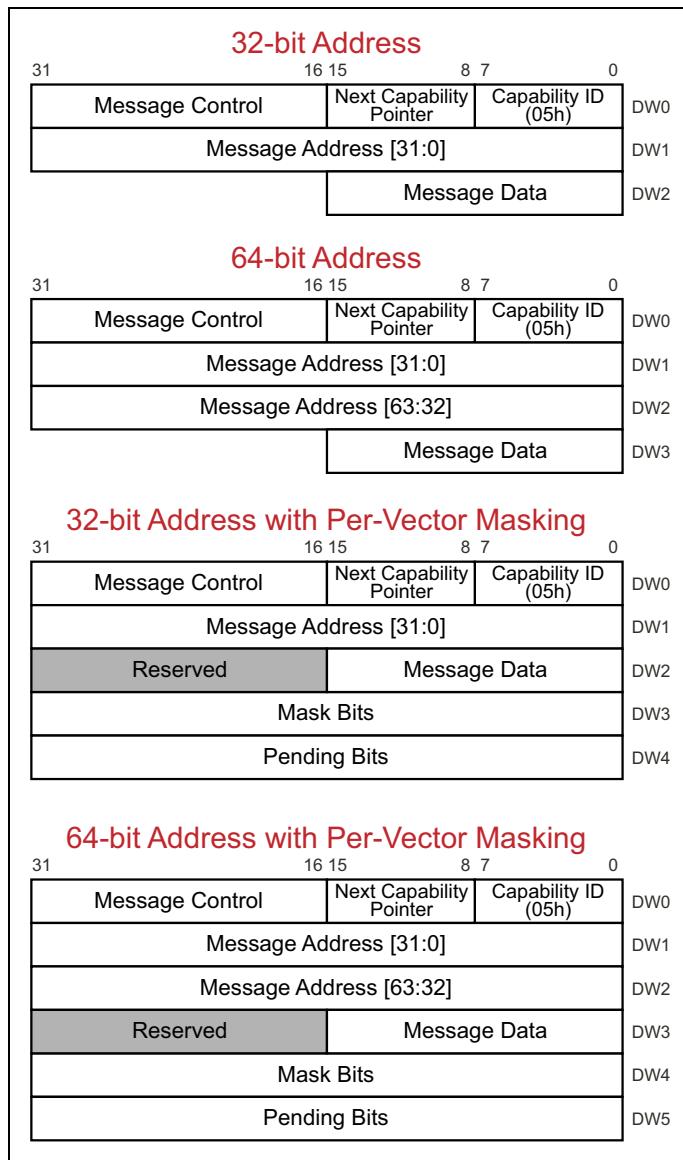
## The MSI Capability Structure

The MSI Capability Structure resides in the PCI-compatible config space area (first 256 bytes). There are four variations of the MSI Capability Structure based on whether it supports 64-bit addressing or only 32-bit and whether it supports

# Chapter 17: Interrupt Support

per vector masking or not. Native PCIe devices are required to support 64-bit addressing. All four variations of the MSI Capability Structure can be found in Figure 17-13 on page 813.

Figure 17-13: MSI Capability Structure Variations



# PCI Express 3.0 Technology

---

## Capability ID

A Capability ID value of **05h** identifies the MSI capability and is a read-only value.

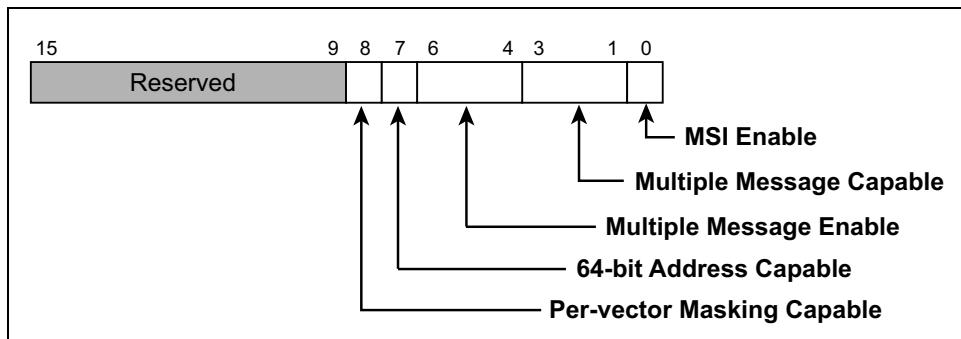
## Next Capability Pointer

The second byte of the register is a read-only value that gives the dword-aligned offset from the top of config space to the next Capability Structure in the linked list of structures or else contains 00h to indicate the end of the linked list.

## Message Control Register

Figure 17-14 on page 814 and Table 17-2 on page 814 illustrate the layout and usage of the Message Control register.

*Figure 17-14: Message Control Register*



*Table 17-2: Format and Usage of Message Control Register*

Bit(s)	Field Name	Description
0	MSI Enable	<p>Read/Write. State after reset is 0, indicating that the device's MSI capability is disabled.</p> <ul style="list-style-type: none"><li>• 0 = Function is <b>disabled</b> from using <b>MSI</b>. It must use MSI-X or else INTx Messages.</li><li>• 1 = Function is <b>enabled</b> to use <b>MSI</b> to request service and won't use MSI-X or INTx Messages.</li></ul>

# Chapter 17: Interrupt Support

---

*Table 17-2: Format and Usage of Message Control Register (Continued)*

<b>Bit(s)</b>	<b>Field Name</b>	<b>Description</b>																		
3:1	Multiple Message Capable	<p>Read-Only. System software reads this field to determine how many messages (interrupt vectors) the Function would like to use. The requested number of messages is a power of two, therefore a Function that would like three messages must request that four messages be allocated to it.</p> <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; width: 15%;"><b>Value</b></th><th style="text-align: left;"><b>Number of Messages Requested</b></th></tr> </thead> <tbody> <tr><td>000b</td><td>1</td></tr> <tr><td>001b</td><td>2</td></tr> <tr><td>010b</td><td>4</td></tr> <tr><td>011b</td><td>8</td></tr> <tr><td>100b</td><td>16</td></tr> <tr><td>101b</td><td>32</td></tr> <tr><td>110b</td><td>Reserved</td></tr> <tr><td>111b</td><td>Reserved</td></tr> </tbody> </table>	<b>Value</b>	<b>Number of Messages Requested</b>	000b	1	001b	2	010b	4	011b	8	100b	16	101b	32	110b	Reserved	111b	Reserved
<b>Value</b>	<b>Number of Messages Requested</b>																			
000b	1																			
001b	2																			
010b	4																			
011b	8																			
100b	16																			
101b	32																			
110b	Reserved																			
111b	Reserved																			
6:4	Multiple Message Enable	<p>Read/Write. After system software reads the Multiple Message Capable field (previous row in this table) to see how many messages (interrupt vectors) are requested by the Function, it programs a 3-bit value in this field indicating the actual number of messages allocated to the Function. The number allocated can be equal to or less than the number actually requested. The state of this field after reset is 000b.</p> <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; width: 15%;"><b>Value</b></th><th style="text-align: left;"><b>Number of Messages Requested</b></th></tr> </thead> <tbody> <tr><td>000b</td><td>1</td></tr> <tr><td>001b</td><td>2</td></tr> <tr><td>010b</td><td>4</td></tr> <tr><td>011b</td><td>8</td></tr> <tr><td>100b</td><td>16</td></tr> <tr><td>101b</td><td>32</td></tr> <tr><td>110b</td><td>Reserved</td></tr> <tr><td>111b</td><td>Reserved</td></tr> </tbody> </table>	<b>Value</b>	<b>Number of Messages Requested</b>	000b	1	001b	2	010b	4	011b	8	100b	16	101b	32	110b	Reserved	111b	Reserved
<b>Value</b>	<b>Number of Messages Requested</b>																			
000b	1																			
001b	2																			
010b	4																			
011b	8																			
100b	16																			
101b	32																			
110b	Reserved																			
111b	Reserved																			

# PCI Express 3.0 Technology

---

Table 17-2: Format and Usage of Message Control Register (Continued)

Bit(s)	Field Name	Description
7	64-bit Address Capable	Read-Only. <ul style="list-style-type: none"><li>• 0 = Function does not implement the upper 32 bits of the Message Address register; only a 32-bit address is possible.</li><li>• 1 = Function implements the upper 32 bits of the Message Address register and is capable of generating a 64-bit memory address.</li></ul>
8	Per-Vector Masking Capable	Read-Only. <ul style="list-style-type: none"><li>• 0 = Function does not implement the Mask Bit register or the Pending Bit register; software does NOT have the ability to mask individual interrupts with this capability structure.</li><li>• 1 = Function does implement the Mask Bit register or the Pending Bit register; software does have the ability to mask individual interrupts with this capability structure.</li></ul>
15:9	Reserved	Read-Only. Always zero.

## Message Address Register

The lower two bits of the 32-bit Message Address register are zero and cannot be changed, forcing the address assigned by software to be dword aligned. Typically, this would be the address of the Local APIC in the system CPU. In x86-based systems (Intel-compatible), this address has traditionally been FEEEx\_xxxxh where the lower 20 bits indicate which Local APIC is being targeted as well as some other info about the interrupt itself. It is important to note that how the address is interpreted is platform specific and is not dictated in the PCI or PCIe specs.

The register containing bits [63:32] of the Message Address are required for native PCI Express devices but is optional for legacy endpoints. This register is present if Bit 7 of the Message Control register is set. If so, it is a read/write register used in conjunction with the Message Address [31:0] register to enable a 64-bit memory address for interrupt delivery from this Function.

## Message Data Register

System software writes a base message data pattern into this 16-bit, read/write register. When the Function generates an interrupt request, it writes a 32-bit data value to the memory address specified in the Message Address register. The upper 16 bits of this data are always set to zero, while the lower 16 bits are supplied by the Message Data register.

If more than one message has been assigned to the Function, it modifies the lower bits (the number of modifiable bits depends on how many messages have been assigned to the Function by configuration software) of the Message Data register value to form the appropriate value for the event it wishes to report. As an example, refer to “Basics of Generating an MSI Interrupt Request” on page 820.

## Mask Bits Register and Pending Bits Register

If the Function supports per-vector masking (indicated in bit [8] of the Message Control register) then these registers are present. The max number of interrupt messages (itnerrupt vectors) that can be requested and assigned to a Function using MSI is 32. So these two registers are 32 bits in length with each potential interrupt message having its own mask and pending bit. If bit [0] of the Mask Bits register is set, then interrupt message 0 is masked (this is the base vector from this Function). If bit [1] is set, then interrupt message 1 is masked (this is the base vector + 1).

When an interrupt message is masked, the MSI for that vector cannot be sent. Instead, the corresponding Pending Bit is set. This allows software to mask individual interrupts from a Function and then periodically poll the Function to see if there are any masked interrupts that are pending.

If software clears a mask bit and the corresponding pending bit is set, the Function must send the MSI request at that time. Once the interrupt message has been sent, the Function would clear the pending bit.

---

## Basics of MSI Configuration

The following list specifies the steps taken by software to configure MSI interrupts for a PCI Express device. Refer to Figure 17-15 on page 819.

1. At startup time, enumeration software scans the system for all PCI-compatible Functions (see “Single Root Enumeration Example” on page 109 for a discussion of the enumeration process).

## **PCI Express 3.0 Technology**

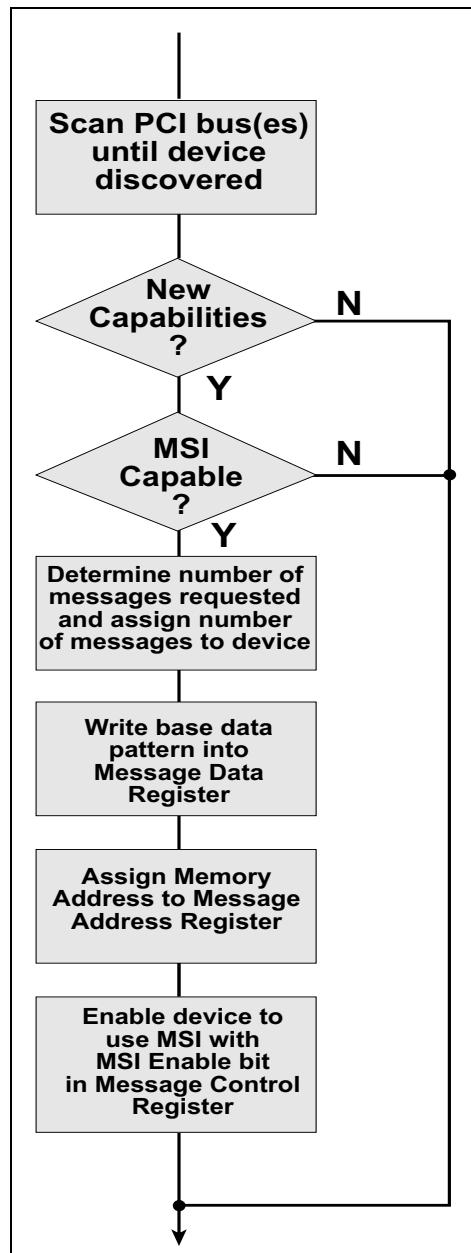
---

2. Once a Function is discovered software reads the Capabilities List Pointer, to find the location of the first capability structure in the linked list.
3. If the MSI Capability structure (Capability ID of 05h) is found in the list, software reads the Multiple Message Capable field in the device's Message Control register to determine how many event-specific messages the device supports and if it supports a 64-bit message address or only 32-bit. Software then allocates a number of messages equal to or less than that and writes that value into the Multiple Message Enable field. At a minimum, one message will be allocated to the device.
4. Software writes the base message data pattern into the device's Message Data register and writes a dword-aligned memory address to the device's Message Address register to serve as the destination address for MSI writes.
5. Finally, software sets the MSI Enable bit in the device's Message Control register, enabling it to generate MSI writes and disabling other interrupt delivery options.

# Chapter 17: Interrupt Support

---

Figure 17-15: Device MSI Configuration Process



## Basics of Generating an MSI Interrupt Request

Figure 17-16 on page 821 illustrates the contents of an MSI Memory Write Transaction Header and Data field. Key points include:

- Format field must be 011b for native functions, indicating a 4DW header (64-bit address) with Data, but it may be 010b for Legacy Endpoints, indicating a 32-bit address.
- The Attribute bits for No Snoop and Relaxed Ordering must be zero.
- Length field must be 01h to indicate maximum data payload of 1DW.
- First BE field must be 1111b, indicating valid data in all four bytes of the DW, even though the upper two bytes will always be zero for MSI.
- Last BE field must be 0000b, indicating a single DW transaction.
- Address fields within the header come directly from the address fields within the MSI Capability registers.
- Lower 16 bits of the Data payload are derived from the data field within the MSI Capability registers.

---

## Multiple Messages

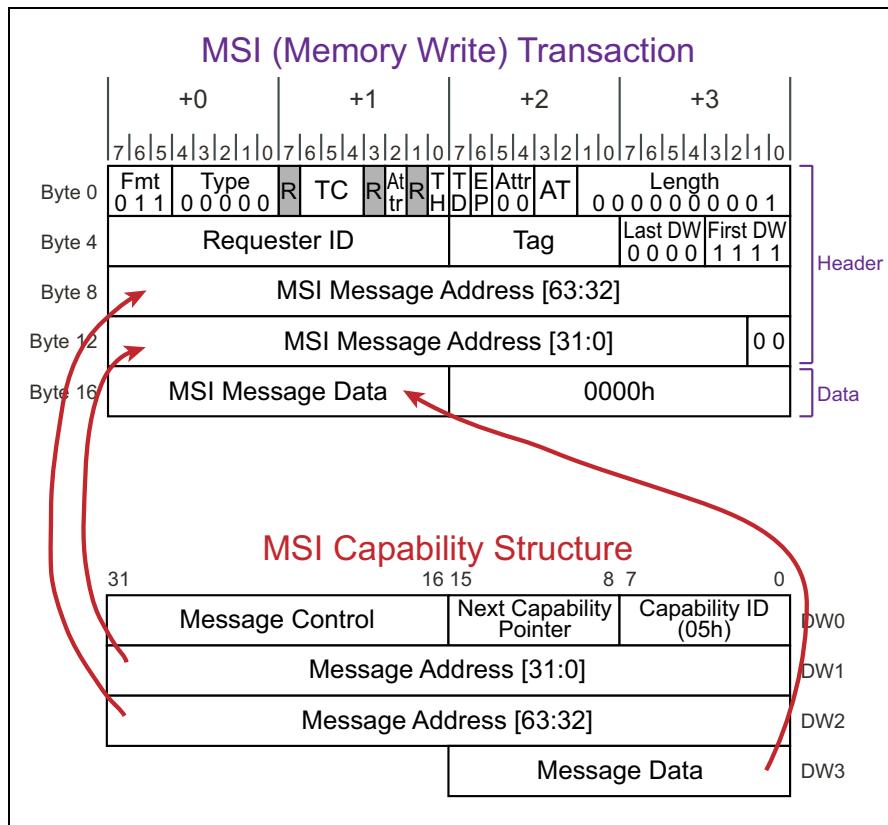
If system software allocated more than one message to the Function, the multiple values are created by modifying the lower bits of the assigned Message Data value to send a different message for each device-specific event type.

As an example, assume the following:

- Four messages have been allocated to a device.
- A data value of 49A0h has been assigned to the device's Message Data register.
- Memory address FEEF\_F00Ch has been written into the device's Message Address register.
- When one of the four events occurs, the device generates a request by performing a dword write to memory address FEEF\_F00Ch with a data value of 0000\_49A0h, 0000\_49A1h, 0000\_49A2h, or 0000\_49A3h. In other words, the lower two bits of the data value are modified to specify which event occurred. If this Function would have been allocated 8 messages, then the lower three bits could be modified. Also, the device always uses 0000h for the upper 2 bytes of its message data value.

# Chapter 17: Interrupt Support

Figure 17-16: Format of Memory Write Transaction for Native-Device MSI Delivery



## The MSI-X Model

### General

The 3.0 revision of the PCI spec added support for MSI-X, which has its own capability structure. MSI-X was motivated by a desire to alleviate three shortcomings of MSI:

- 32 vectors per function are not enough for some applications.
- Having only one destination address makes static distribution of interrupts across multiple CPUs difficult. The most flexibility would be achieved if a unique address could be assigned for each vector.

# PCI Express 3.0 Technology

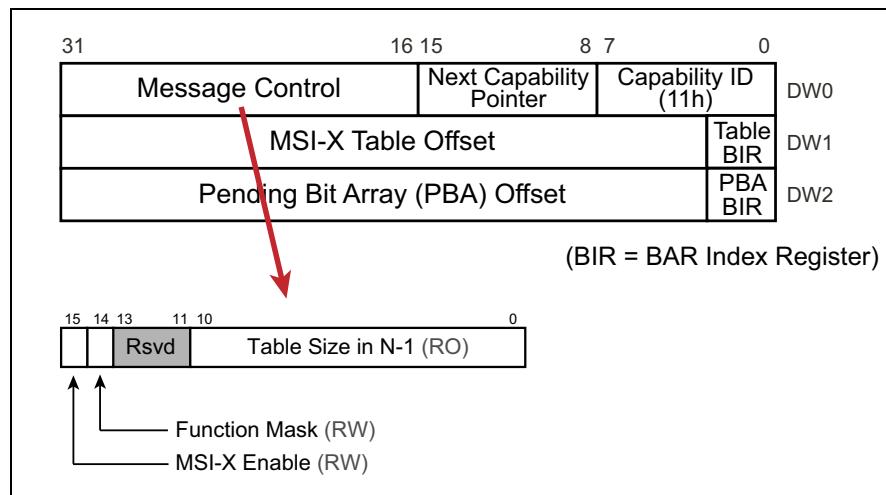
- In several platforms, like x86-based systems, the vector number of the interrupt indicates its priority relative to other interrupts. With MSI, a single Function could be allocated multiple interrupts, but all the interrupt vectors would be contiguous, meaning similar priority. This is not a good solution if some interrupts from this Function should be high priority and others should be low priority. A better approach would be for software to designate a unique vector (message data value), that does not have to be contiguous, for each interrupt allocated to the Function.

Keeping those goals in mind, it's easy to understand the register changes that were implemented to provide more vectors with each vector being assigned a target address and message data value.

## MSI-X Capability Structure

As shown in Figure 17-17 on page 822, the Message Control register is quite different from MSI. Interestingly, even though MSI-X can support up to 2048 vectors per Function versus the 32 for MSI, the number of configuration registers for MSI-X is actually a little smaller than for MSI. That's because the vector information isn't contained here. Instead, it's in a memory location (MMIO) pointed to by the Table BIR (Base address Indicator Register), as shown in Figure 17-18 on page 824.

Figure 17-17: MSI-X Capability Structure



## Chapter 17: Interrupt Support

---

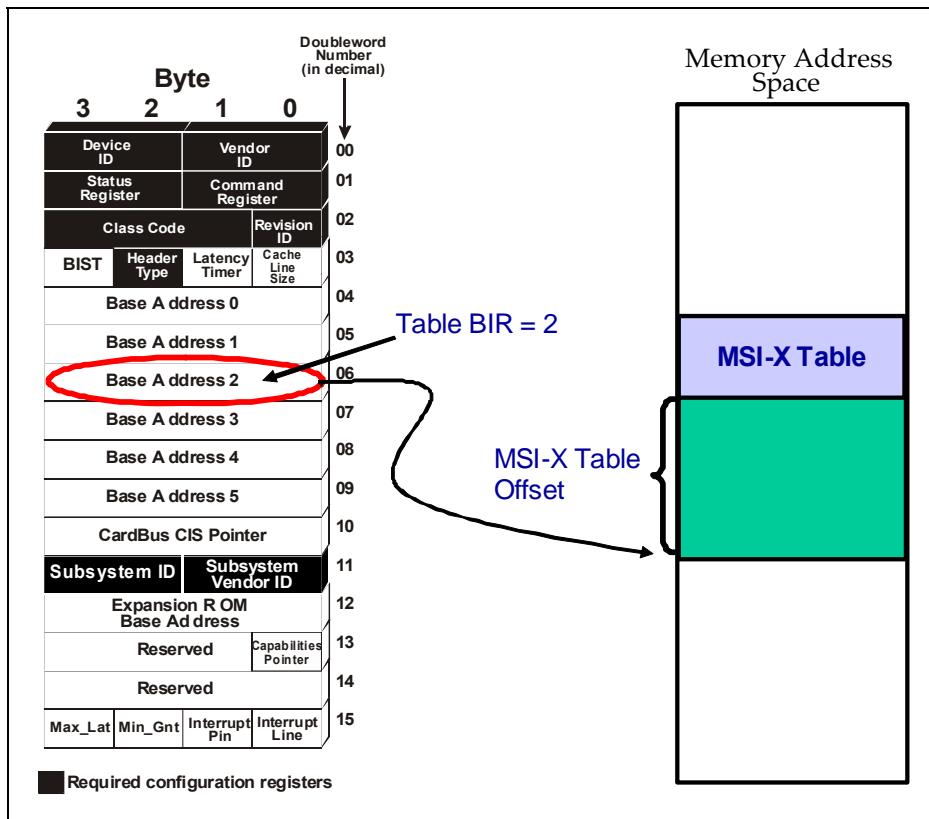
Table 17-3: Format and Usage of MSI-X Message Control Register

Bit(s)	Field Name	Description
10:0	Table Size	Read-Only. This field indicates the number of interrupt messages (vectors) that this Function supports. The value here is interpreted in an N-1 fashion, so a value of 0 means 1 vector. A value of 7 means 8 vectors. Each vector has its own entry in the MSI-X Table and its own bit in the Pending Bit Array.
13:11	Reserved	Read-Only. Always zero.
14	Function Mask	Read/Write. This field provides system software an easy way to mask all the interrupts from a Function. If this bit is cleared, interrupts can still be masked individually by setting the mask bit within each vector's MSI-X table entry.
15	MSI-X Enable	Read/Write. State after reset is 0, indicating that the device's MSI-X capability is disabled. <ul style="list-style-type: none"><li>• <b>0</b> = Function is <b>disabled</b> from using <b>MSI-X</b>. It must use MSI or INTx Messages.</li><li>• <b>1</b> = Function is <b>enabled</b> to use <b>MSI-S</b> to request service and won't use MSI-X or INTx Messages.</li></ul>

# PCI Express 3.0 Technology

---

Figure 17-18: Location of MSI-X Table



---

## MSI-X Table

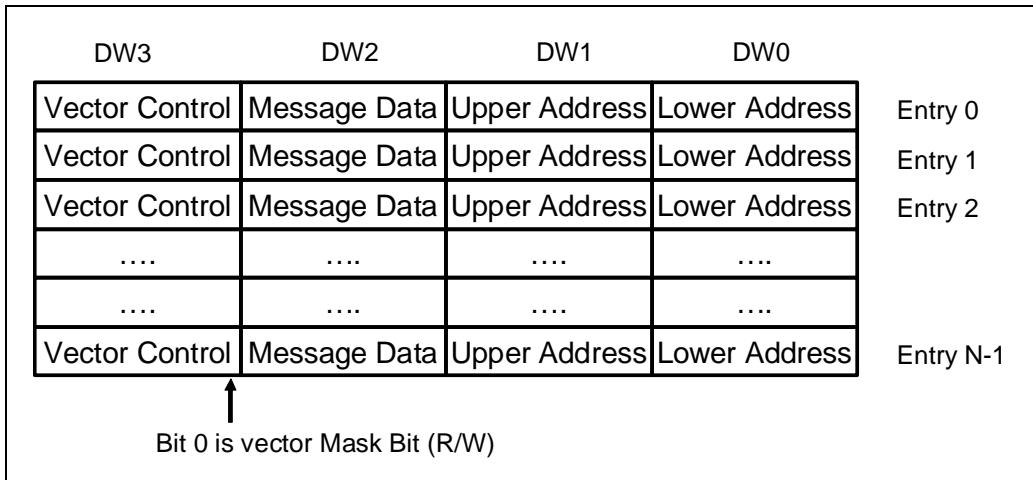
The MSI-X Table itself is an array of vectors and addresses, as shown in Figure 17-19 on page 825. Each entry represents one vector and contains four Dwords. DW0 and DW1 supply a unique 64-bit address for that vector, while DW2 gives a unique 32-bit data pattern for it. DW3 only contains one bit at present: a mask bit for that vector, allowing each vector to be independently masked off as needed.

# Chapter 17: Interrupt Support

Figure 17-19: MSI-X Table Entries

DW3	DW2	DW1	DW0	
Vector Control	Message Data	Upper Address	Lower Address	Entry 0
Vector Control	Message Data	Upper Address	Lower Address	Entry 1
Vector Control	Message Data	Upper Address	Lower Address	Entry 2
....	....	....	....	
....	....	....	....	
Vector Control	Message Data	Upper Address	Lower Address	Entry N-1

Bit 0 is vector Mask Bit (R/W)

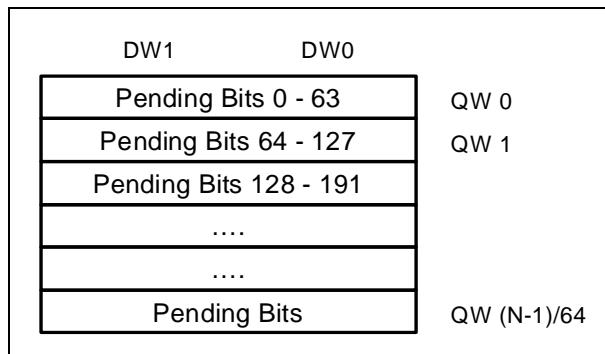


## Pending Bit Array

In much the same way, the Pending Bit Array is also located within a memory address. It can use the same BIR value (same BAR) as the MSI-X Table with a different offset, or it could use a different BAR altogether. The array, shown in Figure 17-20, simply contains a bit for every vector that will be used. If the event to trigger that interrupt occurs but its Mask Bit has been set, then an MSI-X transaction will not be sent. Instead, the corresponding pending bit is set. Later, if that vector is unmasked and the pending bit is still set, the interrupt will be generated at that time.

# PCI Express 3.0 Technology

*Figure 17-20: Pending Bit Array*



---

## **Memory Synchronization When Interrupt Handler Entered**

---

### **The Problem**

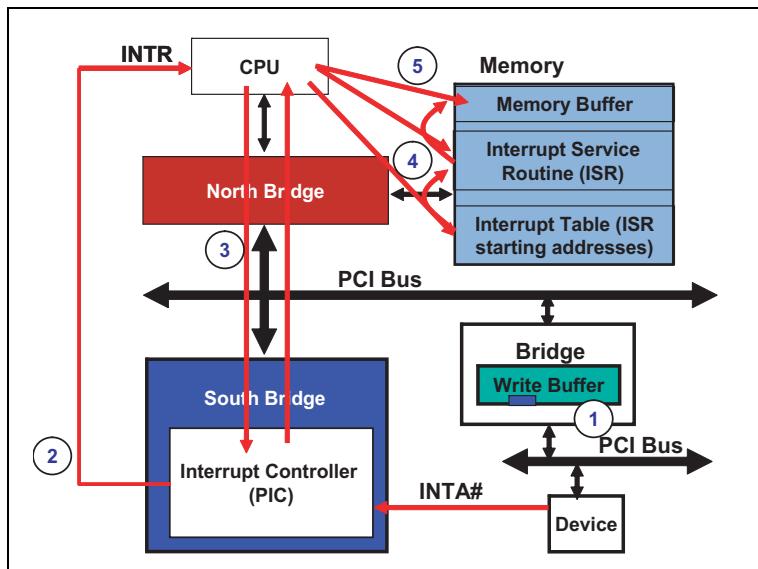
There is a potential problem with any interrupt scheme when data is being delivered. For example, if the device has previously sent data and wants to report that with an interrupt, a unexpected delay on data delivery could allow the interrupt to arrive too soon. That might happen in the bridge data buffer shown in Figure 17-21 on page 827, and the result is a race condition. The steps are similar to our earlier discussion (see “The Legacy Model” on page 796):

1. The function writes a data block toward memory. The write completes on the local bus as a posted transaction, meaning that the sender has finished all it needed to do and the transaction is considered completed.
2. An interrupt is delivered to notify software that some requested data is now present in memory. However, the data has been delayed in the bridge for some reason.
3. The interrupt vector is fetched as before.
4. The ISR starting address is fetched and control is passed to it.
5. The ISR reads from the target memory buffer but the data payload still hasn't been delivered so it fetches stale data, possibly causing an error.

# Chapter 17: Interrupt Support

---

Figure 17-21: Memory Synchronization Problem



---

## One Solution

One way to alleviate this problem takes advantage of PCI transaction ordering rules. If the ISR first sends a read request to the device that initiated the interrupt before it attempts to fetch the data, the resulting read completion will follow the same path back to the CPU that any write data would have taken from that device to get to memory. Transaction ordering rules guarantee that a read result in a bridge cannot pass a posted write going in the same direction, so the end result is that the data will get written into memory before the read result will be allowed to reach the CPU. Therefore, if the ISR waits for the read completion to arrive before proceeding, it can be sure that any data will have been delivered to memory and thus the race condition is avoided. Since the read is basically being used as a data flush mechanism, it isn't necessary for it to return any data. In that case the read can be zero length and the data returned is discarded. For that reason, this type of read is sometimes called a "dummy read."

---

## An MSI Solution

MSI can simplify this process, although there are some requirements for it to work (refer to Figure 17-22 on page 829). If the system allows the device to gen-

# **PCI Express 3.0 Technology**

---

erate its own MSI writes rather than going through an intermediary like an IO APIC, then the following example can take place:

1. The device writes the payload data toward memory and it is absorbed by the write buffer in the bridge.
2. The device believes the data has been delivered and signals an interrupt to notify the CPU. In this case, an MSI is sent and uses the same path as the data. Since both data and MSI appear as memory writes to the bridge, the normal transaction ordering rules will keep them in the correct sequence.
3. The payload data is delivered to memory, freeing the path through the bridge for the MSI write.
4. The MSI write is delivered to the CPU Local APIC and the software now knows that the payload data is available.

---

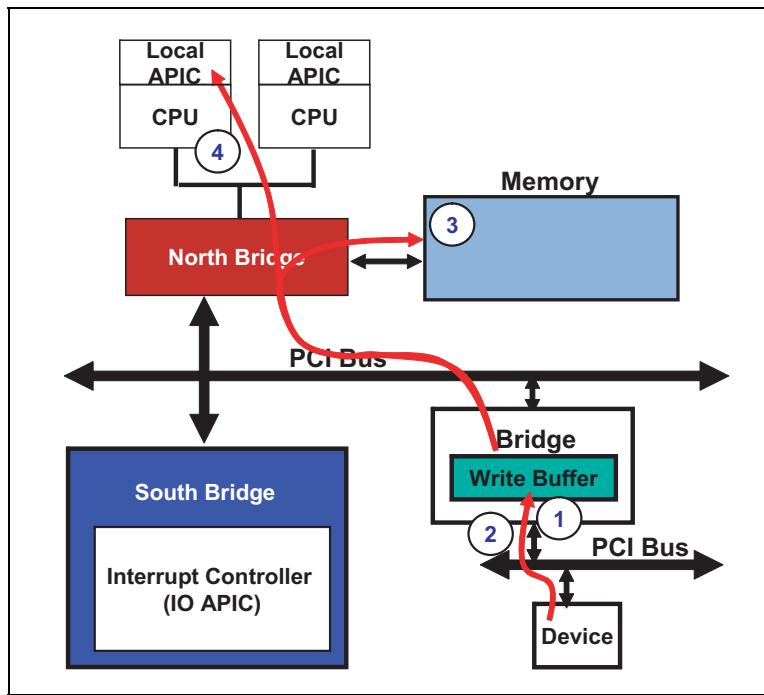
## **Traffic Classes Must Match**

An important point must be stressed here, however. Both the data and MSI must use the same Traffic Class for this to work. Recall that packets that have been assigned different TC values may end up being mapped into different Virtual Channels, and that packets in different VCs have no ordering relationship. If the data were mapped to VC0 and the MSI was mapped to VC1, then the system would be unaware of any ordering relationship between them and unable to enforce memory coherency automatically.

If giving both packets the same TC is not possible, the system would need to use the “dummy read” method instead and the TC of the read request would need to match the TC of the data write packet. It should be clear that even if the same TC is used for both, the use of the Relaxed Ordering bit must be avoided. We’re counting on the transaction ordering rules to achieve memory synchronization, so they must not be relaxed.

# Chapter 17: Interrupt Support

Figure 17-22: MSI Delivery



## Interrupt Latency

The time from signaling an interrupt until software services the device is referred to as the interrupt latency. In spite of its advantages, MSI, like other interrupt delivery mechanisms, does not provide interrupt latency guarantees.

## MSI May Result In Errors

Because MSIs are delivered as Memory Write transactions, an error associated with delivery of an MSI is treated the same as any other Memory Write error condition. See “ECRC Generation and Checking” on page 657 for treatment of ECRC errors, as one example. The concern, of course, is that if an error results in the MSI packet being unrecognized then no interrupt will be seen by the processor. How this condition would be handled is outside the scope of the PCIe spec.

# PCI Express 3.0 Technology

---

## Some MSI Rules and Recommendations

1. It is the intent of the spec that mutually-exclusive messages will be assigned to Functions by system software and that each message will be converted to an exclusive interrupt on delivery to the processor.
2. More than one MSI capability register set per Function is prohibited.
3. A read of the Message Address register produces undefined results.
4. Reserved registers and bits are read-only and always return zero when read.
5. System software can modify Message Control register bits, but the device itself is prohibited from doing so. In other words, modifying the bits by a "back door" mechanism is not allowed.
6. At a minimum, a single message will be assigned to each device (assuming software supports and plans to use MSI in the system).
7. System software must not write to the upper half of the dword that contains the Message Data register.
8. If the device writes the same message multiple times, only one of those messages is guaranteed to be serviced. If all of them must be serviced, the device must not generate the same message again until the previous one has been serviced.
9. If a device has more than one message assigned, and it writes a series of different messages, it is guaranteed that all of them will be serviced.

---

## Special Consideration for Base System Peripherals

Interrupts may also originate in embedded legacy hardware, such as an IO Controller Hub or Super IO device. Some of the typical legacy devices required in such systems include:

- Serial ports
- Parallel ports
- Keyboard and Mouse Controller
- System Timer
- IDE controllers

These devices typically require a specific IRQ line into a PIC or IO APIC, which allows legacy software to interact with them correctly.

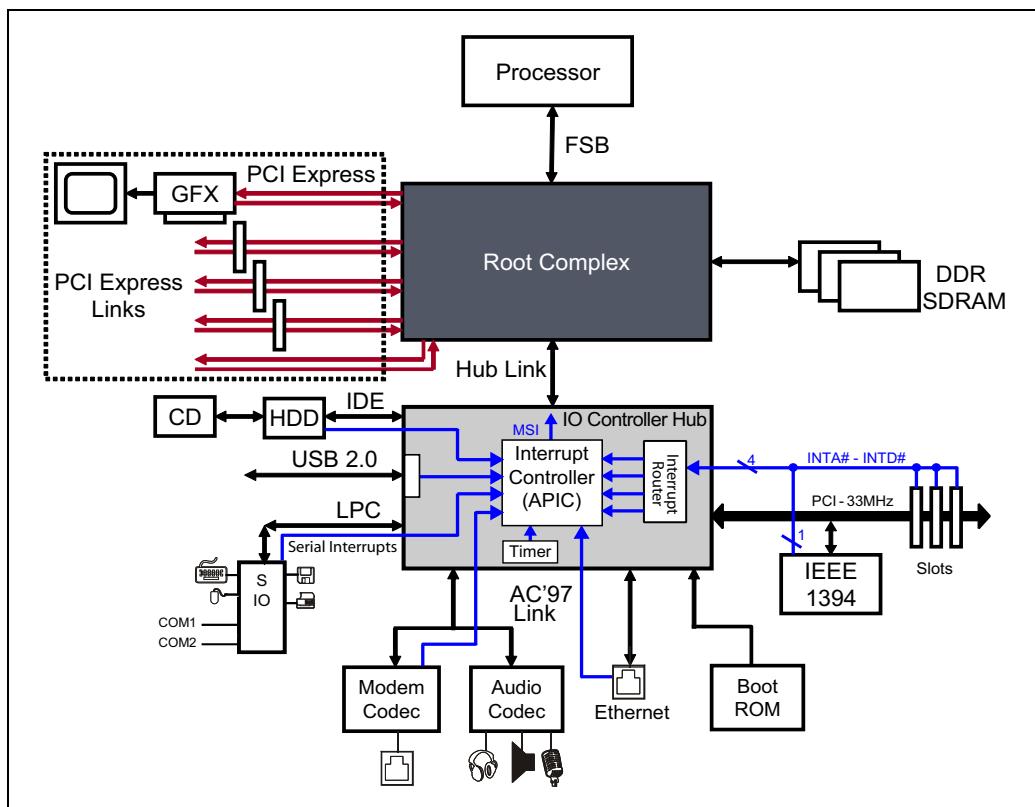
Using the INTx messages does not guarantee that the devices will receive the IRQ assignment they require. The following example illustrates a system that will support the proper legacy interrupt assignment.

## Example Legacy System

Figure 17-23 on page 831 shows a older PCI Express system that includes an IO Controller Hub (ICH) attached to the Root Complex via a proprietary Hub link. The IO APIC embedded within the ICH can generate an MSI when it receives an interrupt request at its inputs. In such an implementation, software can assign the legacy vector number to each input to ensure that the correct legacy software will be called.

The advantage of this approach is that existing hardware can be used to support the legacy requirements of a PCIe platform. This system also requires that the MSI subsystem be configured for use during the boot sequence. The example illustrated eliminates the need for INTx messages unless a PCIe expansion device incorporates a PCI Express-to-PCI Bridge.

Figure 17-23: PCI Express System with PCI-Based IO Controller Hub



## **PCI Express 3.0 Technology**

---

---

---

# 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  $Z_{RX-HIGH-IMP-DC-POS}$  parameter. At 2.5 GT/s, this is no less than  $10\text{ K}\Omega$ . At the higher speeds it must be no less than  $10\text{ K}\Omega$  for voltages below 200mv, and  $20\text{ K}\Omega$  for voltages above 200mv. These are the values when the terminations are not powered.
- Similarly for negative voltages, the  $Z_{RX-HIGH-IMP-DC-NEG}$  parameter, the value is a minimum of  $1\text{ K}\Omega$  in every case.
- Transmitter terminations are required to meet the output impedance  $Z_{TX-DIFF-DC}$  from 80 to  $120\Omega$  for Gen1 and max of  $120\Omega$  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.

# Chapter 18: System Reset

---

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  $Z_{RX-DIFF-DC}$  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

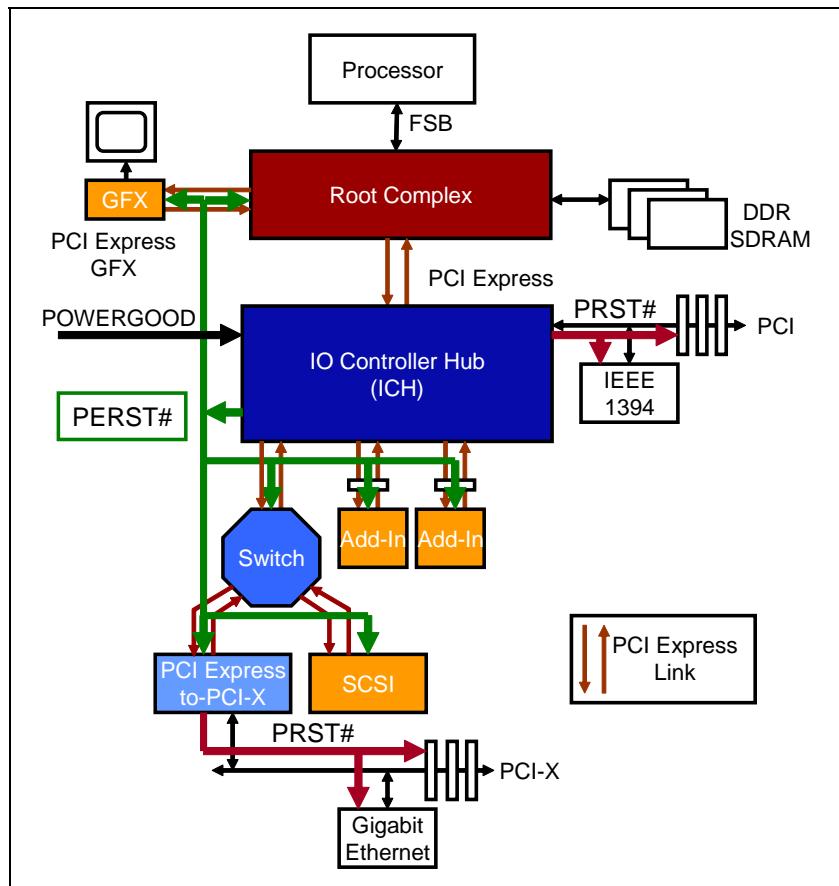
A device must be designed to generate its own reset in hardware upon application of main power. The spec doesn't describe how this would be done, so a self-reset mechanism can be built into the device or added as external logic. For example, an add-in card that detects Power-On may use that event to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

# PCI Express Technology

## 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.

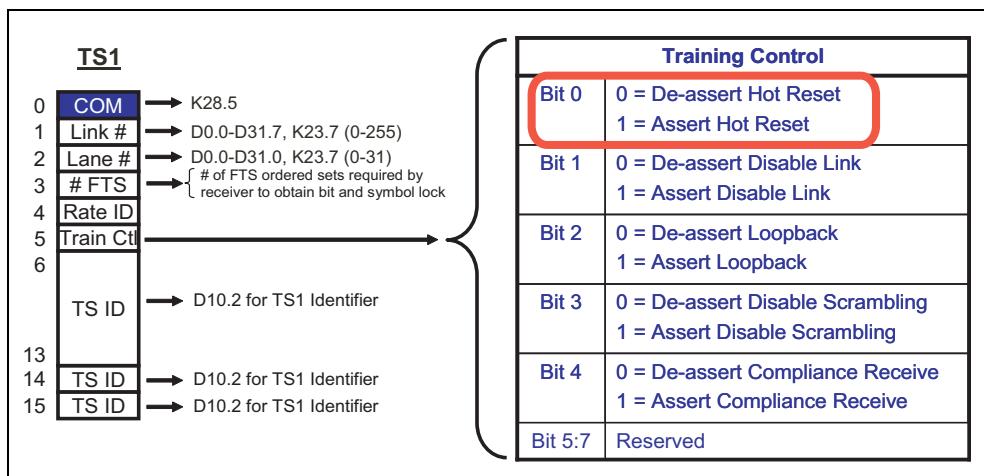
Figure 18-1: PERST# Generation



## Hot Reset (In-band Reset)

A Hot Reset is propagated in-band from one link neighbor to another by sending several TS1s (whose contents are shown in Figure 18-2) with bit 0 of symbol 5 asserted. These TS1s are sent on all Lanes, using the previously negotiated Link and Lane numbers, for 2 ms. Once it's been sent, the Transmitter and Receiver of the Hot Reset will both end up in the Detect LTSSM state (see "Hot Reset State" on page 612).

Figure 18-2: TS1 Ordered-Set Showing the Hot Reset Bit



A hot reset is initiated in software by setting the Secondary Bus Reset bit in a bridge's Bridge Control configuration register, as shown in Figure 18-5 on page 840. Consequently, only devices containing bridges, like the Root Complex or a Switch, can do this. A Switch that receives hot reset on its Upstream Port must broadcast it to all of its Downstream Ports and reset itself. All devices downstream of a switch that receive the hot reset will reset themselves.

## Response to Receiving Hot Reset

- The device's LTSSM goes through the Recovery and Hot Reset state, and then back to the Detect state, where it starts the Link Training process.
- All of the device's state machines, hardware logic, port states and configuration registers (except sticky registers) initialize to their default conditions.

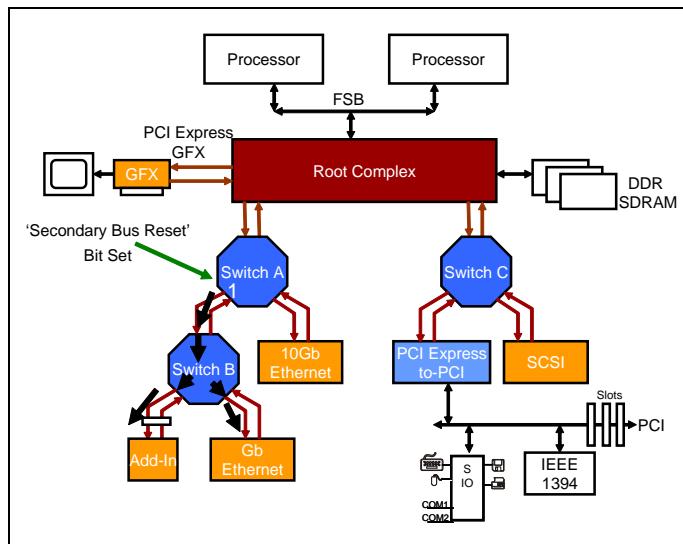
# PCI Express Technology

## Switches Generate Hot Reset on Downstream Ports

A Switch generates a hot reset on all of its Downstream Ports when:

- It receives a hot reset on its Upstream Port
- For a Switch or Bridge Upstream Port, if the Data Link Layer reports a DL\_Down state, the effect is very similar to a hot reset. This can happen when the Upstream Port has lost its connection with an upstream device due to an error that is not recoverable by the Physical Layer or Data Link Layer.
- Software sets the ‘Secondary Bus Reset’ bit of the Bridge Control configuration register associated with the Upstream Port, as shown in Figure 18-3 on page 838.

Figure 18-3: Switch Generates Hot Reset on One Downstream Port



## Bridges Forward Hot Reset to the Secondary Bus

If a bridge such as a PCI Express-to-PCI(-X) bridge detects a hot reset on its Upstream Port, it must assert the PRST# signal on its secondary PCI(-X) bus, as illustrated in Figure 18-4 on page 839.

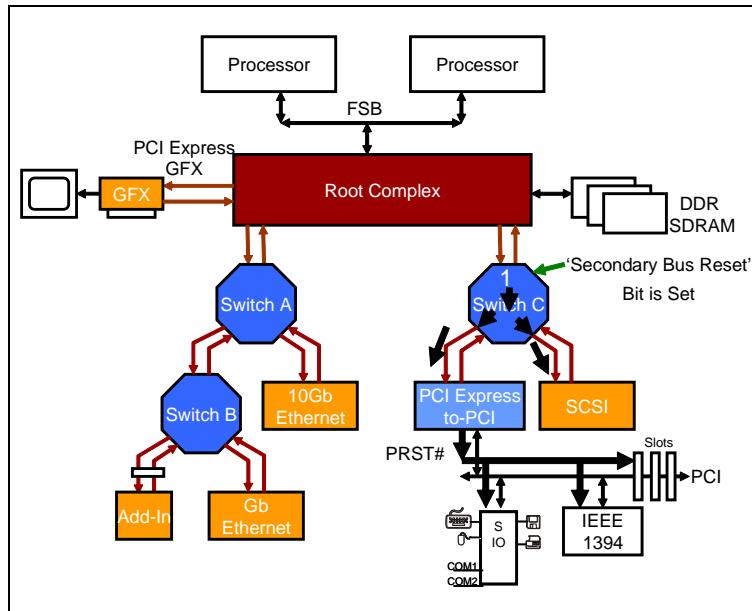
## Software Generation of Hot Reset

Software generates a Hot Reset on a specific port by writing a 1 followed by 0 to the ‘Secondary Bus Reset’ bit in the Bridge Control register of that associated

## Chapter 18: System Reset

port's configuration header (see Figure 18-5 on page 840). Consider the example shown in Figure 18-3 on page 838. Software sets the 'Secondary Bus Reset' register of Switch A's left Downstream Port, causing it to send TS1 Ordered Sets with the Hot Reset bit set. Switch B receives this Hot Reset on its Upstream Port and forwards it to all its Downstream Ports.

Figure 18-4: Switch Generates Hot Reset on All Downstream Ports



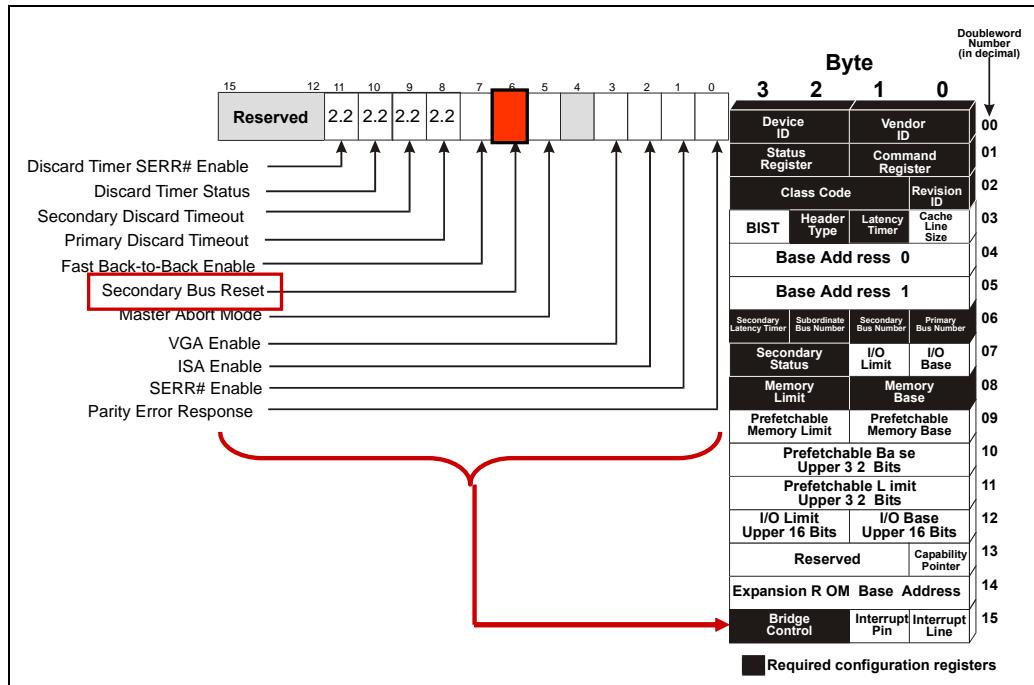
If software sets the Secondary Bus Reset bit of a Switch's Upstream Port, then the switch generates a hot reset on all of its Downstream Ports, as shown in Figure 18-4 on page 839. Here, software sets the Secondary Bus Reset bit in Switch C's Upstream Port, causing it to send TS1s with the Hot Reset bit set on all its Downstream Ports. The PCIe-to-PCI bridge receives this Hot Reset and forwards it on to the PCI bus by asserting PRST#.

Setting the Secondary Bus Reset bit causes a Port's LTSSM to transition to the Recovery state (for more on the LTSSM, see "Overview of LTSSM States" on page 519) where it generates the TS1s with the Hot Reset bit set. The TS1s are generated continuously for 2 ms and then the Port exits to the Detect state where it is ready to start the Link training process.

# PCI Express Technology

The receiver of the Hot Reset TS1s (always downstream) will go to the Recovery state, too. When it sees two consecutive TS1s with the Hot Reset bit set, it goes to the Hot Reset state for a 2ms timeout and then exits to Detect. Both Upstream and Downstream Ports are initialized and end up in the Detect state, ready to begin Link training. If the downstream device is also a Switch or Bridge, it forwards the Hot Reset to its Downstream Ports as well, as shown in Figure 18-3 on page 838.

Figure 18-5: Secondary Bus Reset Register to Generate Hot Reset

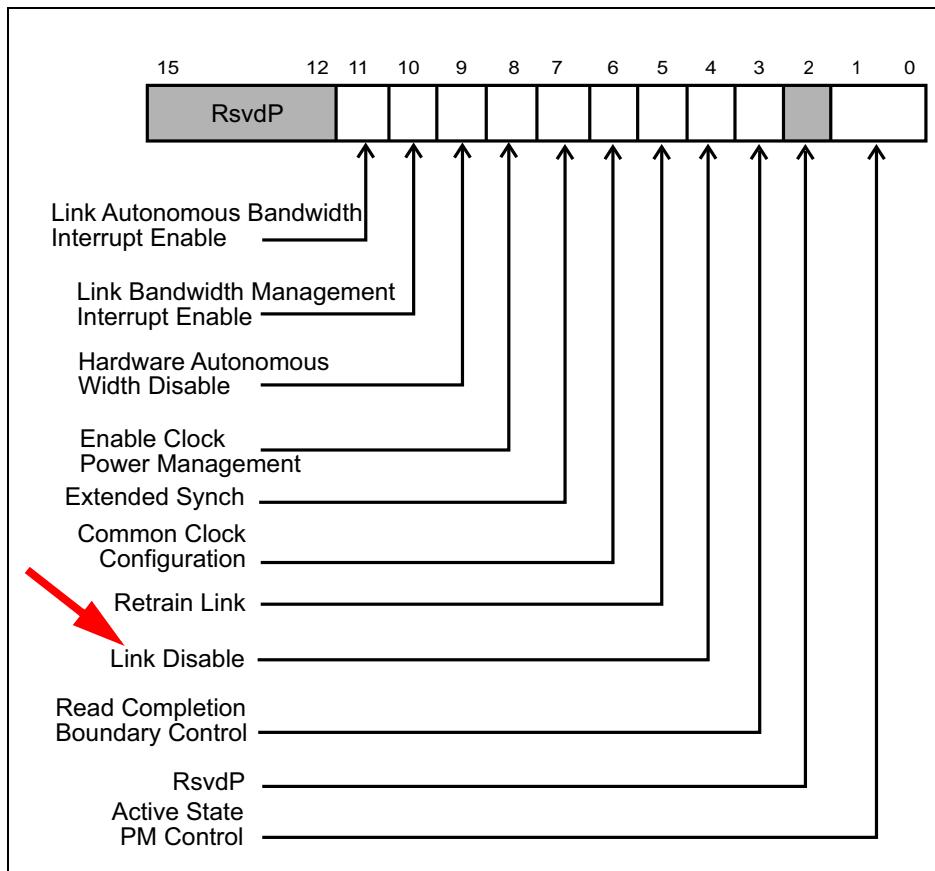


## Software Can Disable the Link

Software can also disable a Link, forcing it to go into Electrical Idle and remain there until further notice. The reason for mentioning that at this point is that disabling the Link also has the effect of causing a Hot Reset on downstream components. Disabling is accomplished by setting the Link Disable bit in the Link Control Register of the Downstream Port, shown in Figure 18-6 on page 841. That causes the Port to go to the Recovery LTSSM state and begin sending TS1s with the Disable bit set. Since this can only be controlled for Downstream Ports if the Link has been disabled, this bit is reserved for Upstream Ports (such as Endpoints or Switch Upstream Ports).

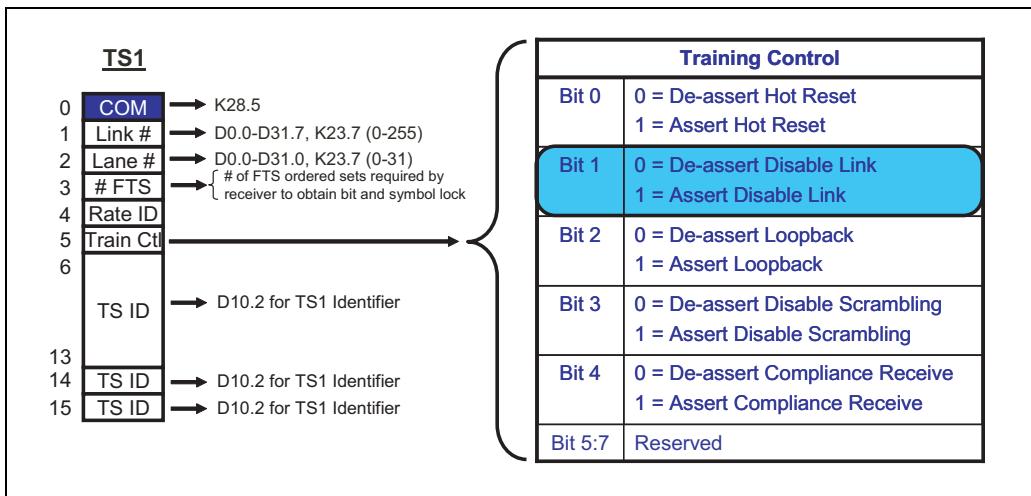
## Chapter 18: System Reset

Figure 18-6: Link Control Register



When the Upstream Port recognizes incoming TS1s with the Disabled bit set, its Physical Layer signals LinkUp=0 (false) to the Link Layer and all the Lanes go to Electrical Idle. After a 2ms timeout, an Upstream Port will go to Detect, but a Downstream Port will remain in the Disabled LTSSM state until directed to exit from it (such as by clearing the Link Disable bit), so the Link will remain disabled and will not attempt training until then.

Figure 18-7: TS1 Ordered-Set Showing Disable Link Bit



---

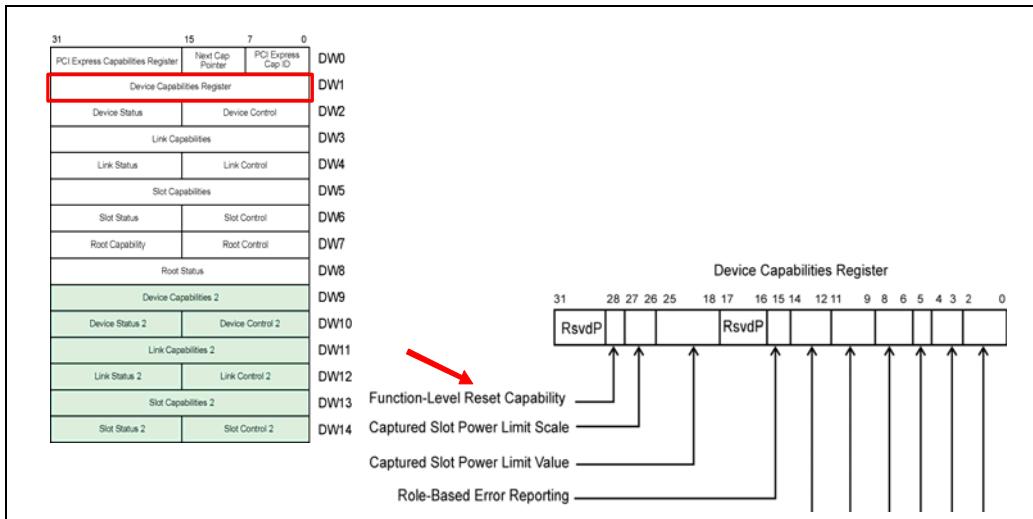
## Function Level Reset (FLR)

The FLR capability allows software to reset just one Function within a multi-function device without affecting the Link that is shared by them all. Its implementation is strongly recommended but isn't required, so software would need to confirm its availability before attempting to use it by examining the Device Capabilities register, as shown in Figure 18-8 on page 843. If the Function-Level Reset Capability bit is set, then an FLR can be initiated by simply setting the Initiate Function-Level Reset bit in the Device Control Register as shown in Figure 18-9 on page 843.

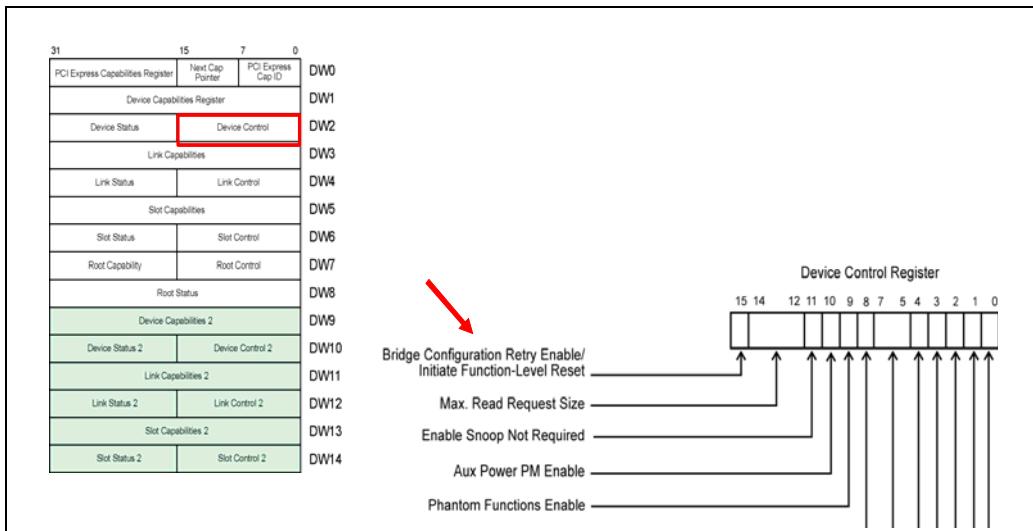
# Chapter 18: System Reset

---

*Figure 18-8: Function-Level Reset Capability*



*Figure 18-9: Function-Level Reset Initiate Bit*



The spec mentions a few examples that motivate the addition of FLR:

1. It can happen that software controlling a Function encounters a problem and is no longer operating correctly. Preventing data corruption necessitates a reset of that Function, but if other Functions within that device are still working properly it would nice to be able to reset just the one having trouble.
2. In a virtualized environment, where applications can migrate from one piece of hardware to another, it's important that when an application is moved off a Function that the Function doesn't retain any information about what it was doing. This prevents information used by one application that might be considered confidential from becoming visible to the new one running on that Function. The simplest way to clean up after migrating the previous application is simply to reset the Function.
3. When software is rebuilding a software stack for a Function, it is sometimes necessary to first put the Function into an uninitialized state. As before, avoiding a reset of all Functions sharing the Link is desirable.

Another feature doesn't appear in the list of cases in the spec but is still a motivating factor in its own right. While a conventional reset will re-initialize everything within the device, it does not require that all external activity, such as traffic on a network interface, must cease right away. FLR adds this requirement and is the only reset that does.

FLR resets the Function's internal state and registers, making it quiescent, but doesn't affect any sticky bits, or hardware-initialized bits, or link-specific registers like Captured Power, ASPM Control, Max\_Payload\_Size or Virtual Channel registers. If an outstanding Assert INTx interrupt message was sent, a corresponding Deassert INTx message must be sent, unless that interrupt was shared by another Function internally that still has it asserted. All external activity for that Function is required to cease when an FLR is received.

---

## Time Allowed

A Function must complete an FLR within 100ms. However, software may need to delay initiating an FLR if there are any outstanding split completions that haven't yet been returned (indicated by the fact that the Transactions Pending bit remains set in the Device Status register). In that case, software must either wait for them to finish before initiating the FLR, or wait 100ms after FLR before attempting to re-initialize the Function. If this isn't managed, a potential data corruption problem arises: a Function may have split transactions outstanding but a reset causes it to lose track of them. If they are returned later they could be

# Chapter 18: System Reset

---

mistaken for responses to new requests that have been issued since the FLR. To avoid this problem, the spec recommends that software should:

1. Coordinate with other software that might access the Function to ensure it doesn't attempt access during the FLR.
2. Clear the entire Command register, thereby quiescing the Function.
3. Ensure that previously-requested Completions have been returned by polling the Transactions Pending bit in the Device Status register until it's cleared or waiting long enough to be sure the Completions won't ever be returned. How long would be long enough? If Completion Timeouts are being used, wait for the timeout period before sending the FLR. If Completion Timeouts are disabled, then wait at least 100ms.
4. Initiate the FLR and wait 100ms.
5. Set up the Function's configuration registers and enable it for normal operation.

When the FLR has completed, regardless of the timing, the Transaction Pending bit must be cleared.

---

## Behavior During FLR

The spec writers chose to describe the behavior of a Function reset in fairly broad terms so as not to preclude any internal steps that designers might wish to take. The following behaviors are listed in the spec:

- The Function must not appear to an external interface as though it was an initialized adapter with an active host. The steps to ensure that all activity on external interfaces is terminated will be design specific. An example would be a network adapter that must not respond to requests that would require an active host during this time.
- The Function must not retain any software-readable state that might include secret information left behind by some previous use of the Function. For example, any internal memory must be cleared or randomized.
- The Function must be configurable as normal by the next driver.
- The Function must return a completion for the configuration write that caused the FLR and then initiate the FLR.

While an FLR is in progress:

- Any requests that arrive are allowed to be silently discarded without logging them or signaling an error. Flow control credits must be updated to maintain the link operation, though.

# PCI Express Technology

---

- Incoming completions can be treated as Unexpected Completions or silently discarded without logging them or signaling an error.
- The FLR itself must be completed within the time described above, but further initialization after that could take longer. If a configuration Request comes in before initialization is completed, the Function must return a completion with CRS (Configuration Retry Status) status. Once a completion is returned with any other status, a CRS status will not be legal again until the Function is reset again.

---

## Reset Exit

After exiting the reset state, Link Training and Initialization must begin within 20 ms. Devices may exit the reset state at different times, since reset signaling is asynchronous, but must begin training within this time.

To allow reset components to perform internal initialization, system software must wait for at least 100 ms from the end of a reset before attempting to send Configuration Requests to them. If software initiates a configuration request to a device after the 100 ms wait time, but the device still hasn't finished its self-initialization, it returns a Completion with status CRS. Since configuration Requests can only be initiated by the CPU, the Completion will be returned to the Root Complex. In response, the Root may re-issue the configuration Request automatically or make the failure visible to software. The spec also states that software should only use 100ms wait periods if CRS Software Visibility has been enabled, since long timeouts or processor stalls may otherwise result.

Devices are allowed a full 1.0 second (-0%/+50%) after a reset before they must give a proper response to a configuration request. Consequently, the system must be careful to wait that long before deciding that an unresponsive device is broken. This value is inherited from PCI and the reason for this lengthy delay may be that some devices implement configuration space as a local memory that must be initialized before it can be seen correctly by configuration software. Its initialization may involve copying the necessary information from a slow serial EEPROM, and so it might take some time.

---

# 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.

# **PCI Express Technology**

---

## **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

# **Chapter 19: Hot Plug and Power Budgeting**

---

- 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**

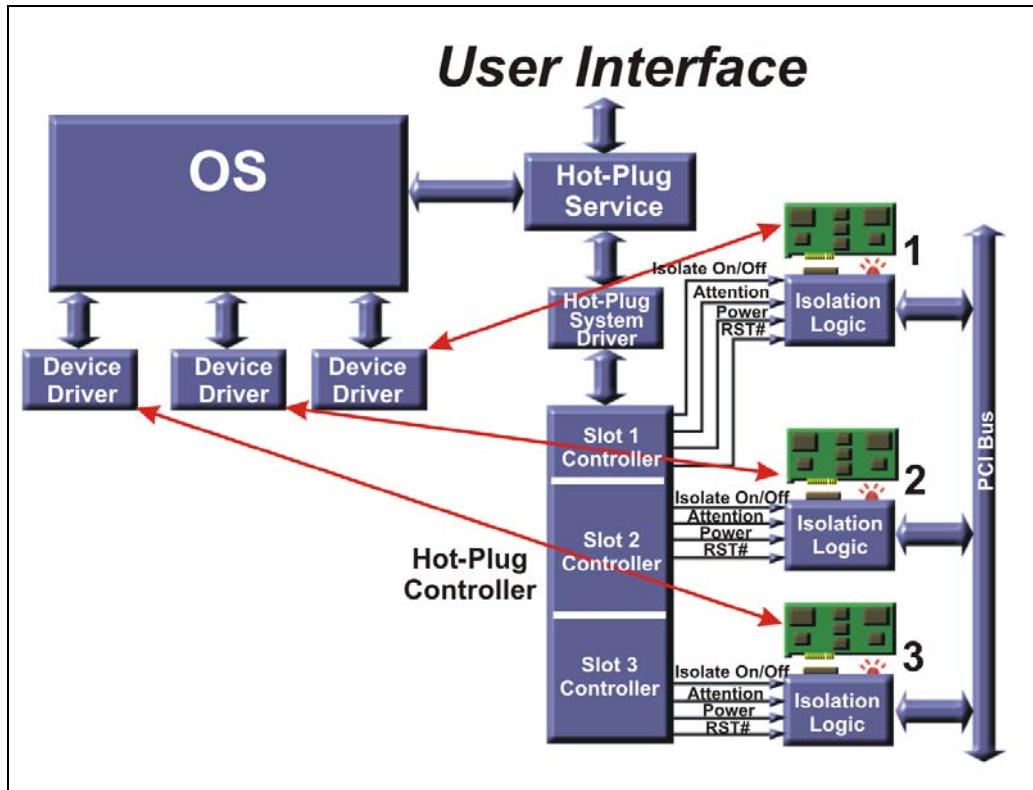
The elements needed to support hot plug are essentially the same in both PCI and PCIe hot plug solutions. Figure 19-1 on page 850 shows the PCI hardware and software elements required to support hot plug. PCI solutions implement a single standardized hot plug controller on the system board that handled all the

# PCI Express Technology

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.

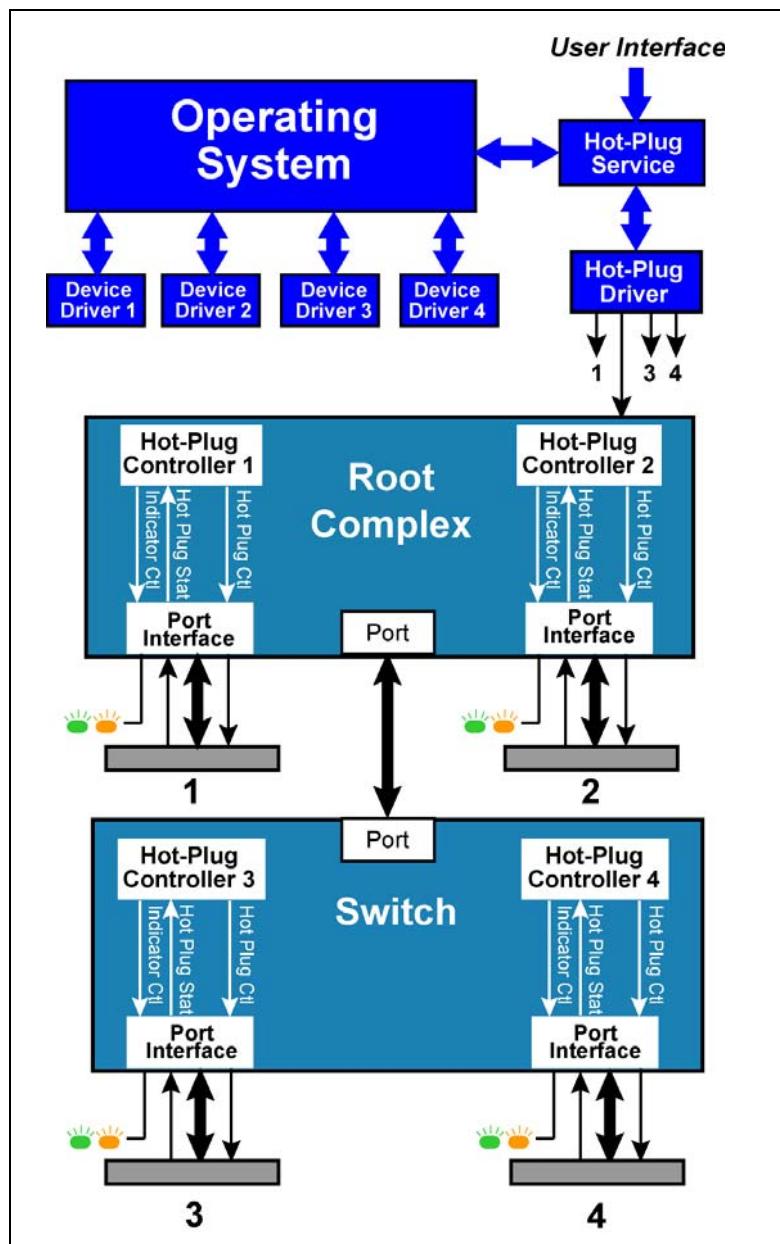
PCIe uses point-to-point connections (see Figure 19-2 on page 851) that eliminate the need for isolation logic but require a separate hot plug controller for each Port to which a connector is attached. A standardized software interface defined for each Root and Switch Port controls hot plug operations.

Figure 19-1: PCI Hot Plug Elements



## Chapter 19: Hot Plug and Power Budgeting

Figure 19-2: PCI Express Hot-Plug Elements



## Elements Required to Support Hot Plug

As shown in Figure 19-2 on page 851 there are several parts involved in making a hot plug environment work. For discussion, let's break these down into software and hardware elements.

### Software Elements

The following table describes the major software elements that support Hot-Plug capability.

*Table 19-1: Introduction to Major Hot-Plug Software Elements*

Software Element	Supplied by	Description
User Interface	OS vendor	An OS-supplied utility that permits the user to request that a connector be powered off to remove a card or turned on to use a card that has just been installed.
Hot-Plug Service	OS vendor	A service that processes requests (referred to as Hot-Plug Primitives) issued by the OS. This includes requests to: <ul style="list-style-type: none"><li>• provide slot identifiers</li><li>• turn card power On or Off</li><li>• turn Attention Indicator On or Off</li><li>• read current power of slot (On or Off)</li></ul> The Hot-Plug Service interacts with the Hot-Plug System Driver to satisfy the requests. The interface (i.e., API) with the Hot-Plug System Driver is defined by the OS vendor.
Standardized Hot-Plug System Driver	System Board vendor or OS	Receives requests (Hot-Plug Primitives) from the Hot-Plug Service within the OS. Interacts with the hardware Hot-Plug Controllers to accomplish requests.

## Chapter 19: Hot Plug and Power Budgeting

---

Table 19-1: Introduction to Major Hot-Plug Software Elements (Continued)

Software Element	Supplied by	Description
Device Driver	Adapter card vendor	<p>Some Hot-Plug-specific capabilities must be incorporated in a Hot-Plug-capable device driver. This includes:</p> <ul style="list-style-type: none"><li>• support for the <b>Quiesce</b> command.</li><li>• optional support of the <b>Pause</b> command.</li><li>• Support for <b>Start</b> command or optional <b>Resume</b> command.</li></ul>

A Hot-Plug-capable system may use an OS that doesn't support Hot-Plug capability. In that case, although the system BIOS would contain Hot-Plug-related software, the Hot-Plug Service would not be present. Assuming that the user doesn't attempt hot insertion or removal of a card, the system will operate as a standard, non-Hot-Plug system:

- The system startup firmware must ensure that all Attention Indicators are Off.
- The spec also states: "the Hot-Plug slots must be in a state that would be appropriate for loading non-Hot-Plug system software."

---

## Hardware Elements

Table 19-2 on page 853 lists the major hardware elements necessary to support PCI Express Hot-Plug operation.

Table 19-2: Major Hot-Plug Hardware Elements

Hardware Element	Description
Hot-Plug Controller	Receives and processes commands issued by the Hot-Plug System Driver. One Controller is associated with each Root or Switch Port that supports hot plug operation. The PCIe spec defines a standard software interface for the Hot-Plug Controller.

# PCI Express Technology

---

Table 19-2: Major Hot-Plug Hardware Elements (Continued)

Hardware Element	Description
Card Slot Power Switching Logic	Allows power to a slot to be turned on or off under program control. Controlled by the Hot Plug controller under the direction of the Hot-Plug System Driver.
Card Reset Logic	Hot Plug Controller drives the PERST# signal to a specific slot as directed by the Hot-Plug System Driver.
Power Indicator	Indicates whether power is currently active on the connector. Controlled by the Hot Plug logic associated with each port and directed by the Hot Plug System Driver.
Attention Indicator	Draws operator attention to a connector that needs service. Controlled by the Hot Plug logic and directed by the Hot-Plug System Driver.
Attention Button	Pressed by the operator to notify Hot Plug software of a request to change a card.
Card Present Detect Pins	There are two of these: PRSNT1# is located at one end of the card slot and PRSNT2# at the opposite end. These pins are shorter than the others so that they disconnect first when a card is removed. The system board ties PRSNT1# to ground and connects PRSNT2# as an input to the Hot-Plug Controller with a pull-up resistor. Additional PRSNT2# pins are defined for wider connectors to support the insertion and recognition of shorter cards installed into longer connectors. The card itself shorts PRSNT1# to PRSNT2#, so that the PRSNT2# input is high if a card is not physically plugged in or low if it is.

## **Card Removal and Insertion Procedures**

The descriptions of typical card removal and insertion that follow are intended to be introductory in nature. It should be noted that the procedures described in the following sections assume that the OS, rather than the Hot-Plug System Driver, is responsible for configuring a newly-installed device. If the Hot-Plug System Driver has this responsibility, the Hot-Plug Service will call the Hot-Plug System Driver and instruct it to configure the newly-installed device.

---

## **On and Off States**

A slot in the On state has the following characteristics:

- Power is applied to the slot.
- REFCLK is on.
- The link is active or in an Active State Power Management state.
- The PERST# signal is deasserted.

A slot in the Off state has the following characteristics:

- Power to the slot is turned off.
- REFCLK is off.
- The link is inactive. (Driver at the root of switch port is in Hi Z state)
- The PERST# signal is asserted.

### **Turning Slot Off**

Steps required to turn off a slot that is currently in the On state:

1. Deactivate the link. This may involve issuing a EIOS to enter the Hi Z state.
2. Assert the PERST# signal to the slot.
3. Turn off REFCLK to the slot.
4. Remove power from the slot.

### **Turning Slot On**

Steps to turn on a slot that is currently in the off state:

1. Apply power to the slot.
2. Turn on REFCLK to the slot

# PCI Express Technology

---

3. Deassert the PERST# signal to the slot. The system must meet the setup and hold timing requirements (specified in the PCI Express spec) relative to the rising edge of PERST#.

Once power and clock have been restored and PERST# removed, the physical layers at both ports will perform link training and initialization. When the link is active, the devices will initialize VC0 (including flow control), making the link ready to transfer TLPs.

---

## Card Removal Procedure

When a card is to be removed, a number of steps are needed to prepare software and hardware for safe removal of the card, and set the indicators for the card being processed. The condition of the indicators during normal operation are:

- Attention Indicator (Amber or Yellow) — “Off” during normal operation.
- Power Indicator (Green) — “On” during normal operation

Software sends requests to the Hot Plug Controller using configuration writes that target the Slot Control Registers implemented by Hot-Plug capable ports. These control the power to the slot and the state of the indicators.

The sequence of events is as follows:

1. The operator requests card removal by pressing the slot’s attention button or by using the system’s user interface to select the Physical Slot number of the card to be removed. If the button was used, the Hot-Plug Controller detects this event and delivers an interrupt to the root complex. The interrupt directs the Hot Plug service to call the Hot Plug System Driver to read slot status information and detect the Attention Button request.
2. Next, the Hot-Plug Service commands the Hot-Plug System Driver to blink the slot’s Power Indicator as visual feedback to the operator for 5 seconds. If this was initiated by pressing the Attention button, the operator can press the button a second time to cancel the request during this 5-second interval.
3. The Power Indicator continues to blink while the Hot Plug software validates the request. If the card is currently in use for some critical system operation, software may deny the request. In that case, it will issue a command to the Hot Plug controller to turn the Power Indicator back ON. The spec also recommends that software notify the operator, perhaps with a message or by logging an entry indicating the reason the request was denied.

## **Chapter 19: Hot Plug and Power Budgeting**

---

4. If the request is validated, the Hot-Plug Service utility commands the card's device driver to quiesce the device. That is, disable its ability to generate new Requests and complete or terminate all outstanding Root or Switch Port requests.
5. Software then issues a command to disable the card's Link via the Link Control register in the Root or Switch Port to which the slot is attached.
6. Next, software commands the Hot Plug Controller to turn the slot off.
7. Following successful power down, software issues the Power Indicator Off Request to turn off the power indicator so the operator knows the card may be removed.
8. The operator releases the Mechanical Retention Latch, if there is one, causing the Hot Plug Controller to remove all switched signals from the slot (e.g., SMBus and JTAG signals). The card can now be removed.
9. The OS deallocates the memory space, IO space, interrupt line, etc. that had been assigned to the device and makes these resources available for assignment to other devices in the future.

---

### **Card Insertion Procedure**

The procedure for installing a new card basically reverses the steps listed for card removal. The following steps assume that the slot was left in the same state that it was in immediately after a card was removed from the connector (in other words, the Power Indicator is in the Off state, indicating the slot is ready for card insertion).

The steps taken to Insert and enable a card are as follows:

1. The operator installs the card and secures the MRL. If implemented, the MRL sensor will signal the Hot-Plug Controller that the latch is closed, causing switched auxiliary signals and  $V_{aux}$  to be connected to the slot.
2. Next, the operator notifies the Hot-Plug Service that the card has been installed by pressing the Attention Button or using the Hot Plug Utility program to select the slot.
3. If the button was pressed, it signals the Hot Plug controller of the event, resulting in status register bits being set and causing a system interrupt to be sent to the Root Complex. Subsequently, Hot Plug software reads slot status from the port and recognizes the request.
4. The Hot-Plug Service issues a request to the Hot-Plug System Driver commanding the Hot Plug Controller to blink the slot's Power Indicator to inform the operator that the card must not be removed. The operator is granted a 5 second abort interval, from the time that the indicators starts to blink, to abort the request by pressing the button a second time.

# PCI Express Technology

---

5. The Power Indicator continues to blink while Hot Plug software validates the request. Note that software may fail to validate the request (e.g., the security policy settings may prohibit the slot being enabled). If the request is not validated, software will issue a command to the Hot Plug controller to turn the Power Indicator back OFF. The spec recommends that software notify the operator via a message or by logging an entry indicating the cause of the request denial.
6. The Hot-Plug Service issues a request to the Hot-Plug System Driver commanding the Hot Plug Controller to turn the slot on.
7. Once power is applied, software issues a command to turn the Power Indicator ON.
8. Once link training is complete, the OS commands the Platform Configuration Routine to configure the card function(s) by assigning the necessary resources.
9. The OS locates the appropriate driver(s) (using the Vendor ID and Device ID, or the Class Code, or the Subsystem Vendor ID and Subsystem ID configuration register values as search criteria) for the function(s) within the PCI Express device and loads it (or them) into memory.
10. The OS then calls the driver's initialization code entry point, causing the processor to execute the driver's initialization code. This code finishes the setup of the device and then sets the appropriate bits in the device's PCI configuration Command register to enable the device.

---

## Standardized Usage Model

---

### Background

Systems based on the original 1.0 version of the PCI Hot Plug spec implemented hardware and software designs that varied widely because the spec did not define standardized registers or user interfaces. Consequently, customers who purchased Hot Plug capable systems from different vendors were confronted with a wide variation in user interfaces that required retraining operators when new systems were purchased. Furthermore, every board designer was required to write software to manage their implementation-specific hot plug controller. The 1.1 revision of the PCI Hot-Plug Controller (HPC) spec defines:

- a standard user interface that eliminates retraining of operators
- a standard programming interface for the hot plug controller, which permits a standardized hot plug driver to be incorporated into the operating system. PCI Express implements registers not defined by the HPC spec,

# **Chapter 19: Hot Plug and Power Budgeting**

---

hence the standard Hot Plug Controller driver implementations for PCI and PCI Express are slightly different.

---

## **Standard User Interface**

The user interface includes the following features:

- Attention Indicator — shows the attention state of the slot with an LED that is on, off, or blinking. The spec defines the blinking frequency as 1 to 2 Hz and 50% (+/- 5%) duty cycle. The state of this indicator is strictly under software control.
- Power Indicator (called Slot State Indicator in PCI HP 1.1) — shows the power status of the slot and also can be on, off, or blinking (at 1 to 2 Hz and 50% (+/- 5%) duty cycle). This indicator is controlled by software; however, the spec permits an exception in the event of a hardware power fault condition.
- Manually Operated Retention Latch and Optional Sensor — secures card within slot and notifies the system when the latch is released
- Electromechanical Interlock (optional) — locks the card or retention latch to prevent the card from being removed while power is applied.
- Software User Interface — allows operator to request hot plug operation
- Attention Button — allows operator to manually request hot plug operation.
- Slot Numbering Identification — provides visual identification of slot on the board.

### **Attention Indicator**

As mentioned in the previous section, the spec requires the system vendor to include an Attention Indicator associated with each Hot-Plug slot. This indicator must be located in close proximity to the corresponding slot and is yellow or amber in color. This Indicator draws the attention of the end user to the slot for service. The spec makes a clear distinction between operational and validation errors and does not permit the attention indicator to report validation errors. Validation errors are problems detected and reported by software prior to beginning the hot plug operation. The behavior of the Attention Indicator is listed in Table 19-3 on page 860.

*Table 19-3: Behavior and Meaning of the Slot Attention Indicator*

Indicator Behavior	Attention State
Off	Normal — Normal Operation
On	Attention — Hot Plug Operation Failed due to an operational problem (e.g., problems with external cabling, add-in cards, software drivers, and power faults)
Blinking	Locate — Slot is being identified at operator's request

## Power Indicator

The power indicator simply reflects the state of main power at the slot, and is controlled by Hot Plug software. The color of this indicator is green and is illuminated when power to the slot is “on.”

The spec specifically prohibits Root or Switch Port hardware from changing the power indicator state autonomously as a result of power fault or other events. A single exception to this rule allows a platform to detect stuck-on power faults. A stuck-on fault is simply a condition in which commands issued to remove slot power are ineffective. If the system is designed to detect this condition the system may override the Root or Switch Port’s command to turn the power indicator off and force it to remain on. This notifies the operator that the card should not be removed from the slot. The spec further states that supporting stuck-on faults is optional and, if handled via system software, “the platform vendor must ensure that this optional feature of the Standard Usage Model is addressed via other software, platform documentation, or by other means.”

The behavior of the power indicator and the related power states are listed in Table 19-4 on page 861. Note that  $V_{aux}$  remains on and switch signals are still connected until the retention latch is released or when the card is removed as detected by the Prsnt1# and Prsnt2# signals.

# **Chapter 19: Hot Plug and Power Budgeting**

---

*Table 19-4: Behavior and Meaning of the Power Indicator*

<b>Indicator Behavior</b>	<b>Power State</b>
Off	Power Off — it is safe to remove or insert a card. All power has been removed as required for hot plug operation. Vaux is only removed when the Manual Retention Latch is released.
On	Power On — removal or insertion of a card is not allowed. Power is currently applied to the slot.
Blinking	Power Transition — card removal or insertion is not allowed. This state notifies the operator that software is currently removing or applying slot power in response to a hot plug request.

## **Manually Operated Retention Latch and Sensor**

The Manual Retention Latch (MRL) is required and holds PCI Express cards rigidly in the slot. Each MRL can implement an optional sensor that notifies the Hot-Plug Controller that the latch has been closed or opened. The spec also allows a single latch that can hold down multiple cards. Such implementations do not support the MRL sensor.

An MRL Sensor is a switch, optical device, or other type of sensor that reports whether the latch is closed or open. If an unexpected latch release is detected, the port automatically disables the slot and notifies system software, although changing the state of the Power or Attention indicators autonomously is not allowed.

The switched signals and auxiliary power (Vaux) must be automatically removed from the slot when the MRL Sensor indicates that the MRL is open, and they must be restored to the slot when the MRL Sensor indicates that the latch is closed. The switched signals are V<sub>aux</sub>, SMBCLK, and SMBDAT.

The spec also describes an alternate method for removing V<sub>aux</sub> and SMBus power when an MRL sensor is not present. The PRSNT#2 pin indicates whether a card is physically installed into the slot and can be used to trigger the port to remove the switched signals.

## Electromechanical Interlock (optional)

The optional electromechanical card interlock mechanism provides a more sophisticated method of ensuring that a card is not removed while power is applied to the slot. The spec does not define the specific nature of the interlock, but states that it can physically lock the add-in card or the MRL in place.

The lock mechanism is controlled via software; however, there is no specific programming interface defined for it. Instead, an interlock is controlled by the same Port signal that enables main power to the slot.

## Software User Interface

An operator may use a software interface to request card removal or insertion. This interface is provided by system software, which also monitors slots and reports status information to the operator. The spec states that the user interface is implemented by the Operating System and consequently is beyond the scope of the spec.

The operator must be able to initiate operations at each slot independent of other slots. Consequently, the operator may initiate a hot-plug operation on one slot using the software user interface or attention button while a hot-plug operation on another slot is in process. This can be done regardless of which interface the operator used to start the first Hot-Plug operation.

## Attention Button

The Attention Button is a momentary-contact push-button switch, located near the corresponding Hot-Plug slot or on a module. The operator presses this button to initiate a hot-plug operation for this slot (e.g., card removal or insertion). Once the Attention Button is pressed, the Power Indicator starts to blink. From the time the blinking begins the operator has 5 seconds to abort the Hot Plug operation by pressing the button a second time.

The spec recommends that if an operation initiated by an Attention Button fails, the system software should notify the operator of the failure. For example, a message explaining the nature of the failure can be reported or logged.

## Slot Numbering Identification

Software and operators must be able to identify a physical slot based on its slot number. Each hot-plug capable port must implement registers that software uses to identify the physical slot number. The registers include a Physical Slot

## **Chapter 19: Hot Plug and Power Budgeting**

---

number and a chassis number. The main chassis is always labeled chassis 0. The chassis numbers for other chassis must be non-zero and are assigned via the PCI-to-PCI bridge's Chassis Number register.

---

### **Standard Hot Plug Controller Signaling Interface**

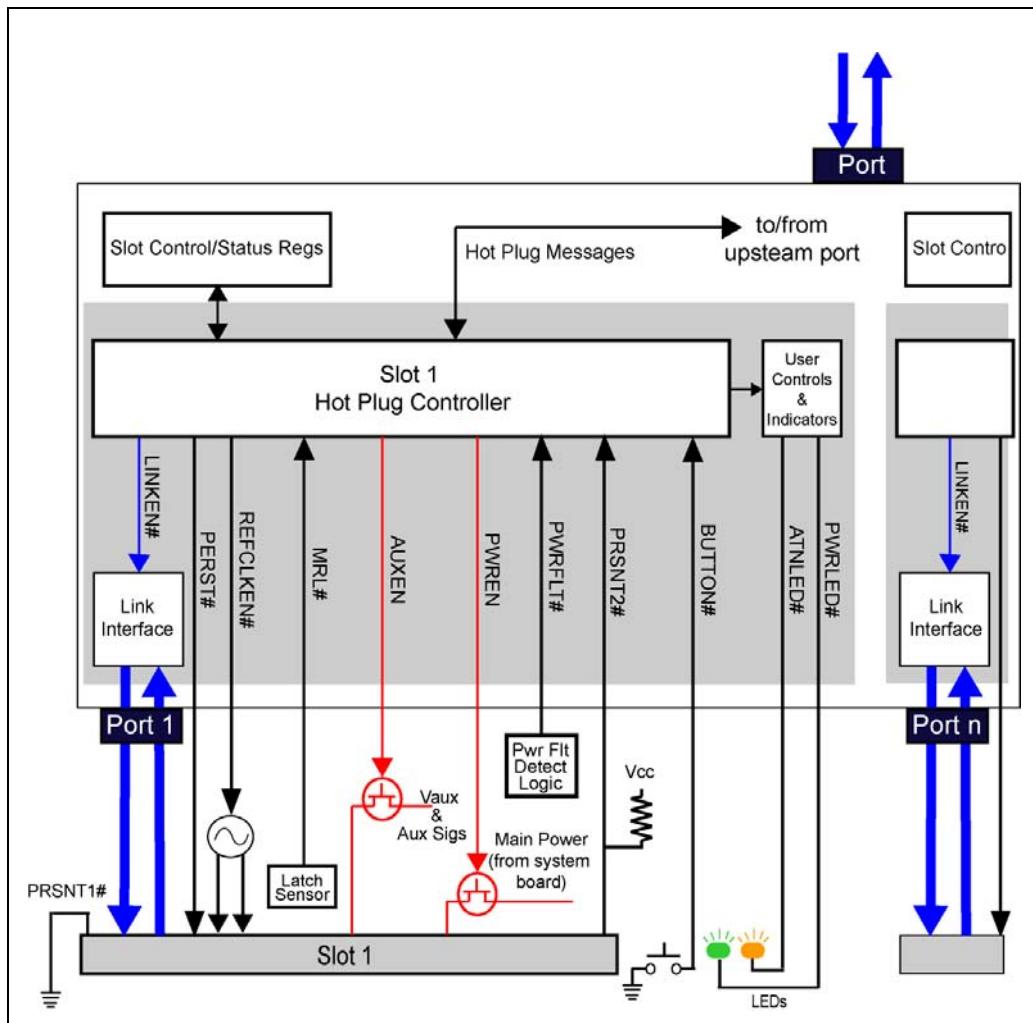
Figure 19-3 on page 864 presents a more detailed view of the logic within Switch Ports, along with the signals routed between the slot and the Port. The importance of the standardized Hot Plug Controller is the common software interface that allows the device driver to be integrated into operating systems.

The PCIe spec, together with the Card ElectroMechanical (CEM) spec, defines the slot signals and the support required for Hot Plug PCI Express. Following is a list of required and optional port interface signals needed to support the Standard Usage Model:

- PWRLED# (required) — port output that controls state of Power Indicator
- ATNLED# (required) — port output controls state of Attention Indicator
- PWREN (required if reference clock is implemented) — port output that controls main power to slot
- REFCLKEN# (required) — port output that controls delivery of reference clock to the slot
- PERST# (required) — port output that controls PERST# at slot
- PRSNT1# (required) — Grounded at the connector
- PRSNT2# (required) — port input, pulled up on system board, that indicates presence of card in slot.
- PWRFLT# (required) — port input that notifies the Hot-Plug controller of a power fault condition detected by external logic
- AUXEN# (required if AUX power is implemented) — port output that controls switched AUX signals and AUX power to slot when MRL is opened and closed. The MRL# signal is required with AUX power is present.
- MRL# (required if MRL Sensor is implemented) — port input from the MRL sensor
- BUTTON# (required if Attention Button is implemented) — port input indicating operator has pressed the Attention Button.

# PCI Express Technology

Figure 19-3: Hot Plug Control Functions within a Switch



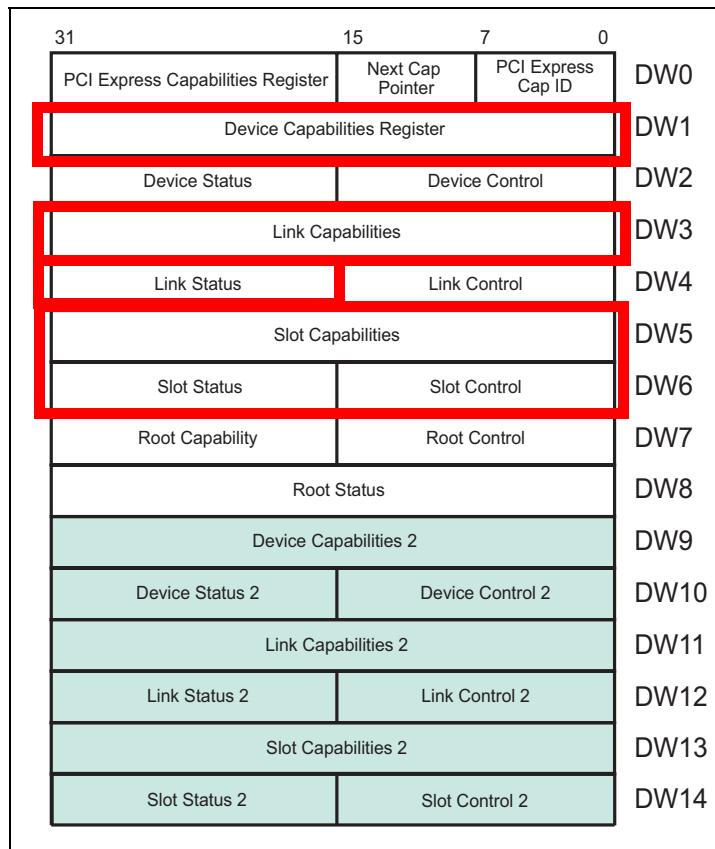
## The Hot-Plug Controller Programming Interface

The standard programming interface to the Hot-Plug Controller is provided via the PCI Express Capability register block, shown in Figure 19-4 on page 865, where the Hot-Plug related registers are highlighted. Hot Plug features are pri-

# Chapter 19: Hot Plug and Power Budgeting

marily found in the Slot Registers defined for Root and Switch Ports. The Device Capability register is also used in some implementations as described later in this chapter.

Figure 19-4: PCIe Capability Registers Used for Hot-Plug



## Slot Capabilities

Figure 19-5 on page 866 illustrates the slot capability register and bit fields. Hardware initializes all of these capability register fields to reflect the features implemented by this port. This register applies to both card slots and rack mount implementations, except for the indicators and attention button. Software must read from the device capability register within the module to determine if indicators and attention buttons are implemented. Table 19-5 on page 866 lists and defines the slot capability fields.

# PCI Express Technology

---

Figure 19-5: Slot Capabilities Register

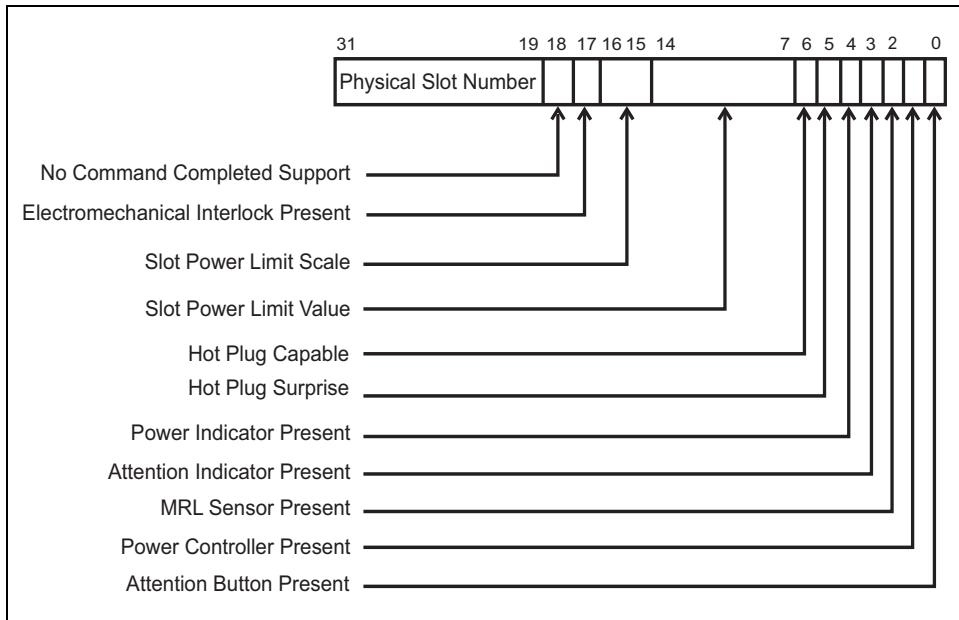


Table 19-5: Slot Capability Register Fields and Descriptions

Bit(s)	Register Name and Description
0	<b>Attention Button Present</b> — indicates the presence of an attention button on the chassis adjacent to the slot.
1	<b>Power Controller Present</b> — indicates the presence of a power controller for this slot.
2	<b>MRL Sensor Present</b> — indicates the presence of a MRL Sensor on the slot.
3	<b>Attention Indicator Present</b> — indicates the presence of an attention indicator on the chassis adjacent to the slot.
4	<b>Power Indicator Present</b> — indicates the presence of a power indicator on the chassis adjacent to the slot.

# Chapter 19: Hot Plug and Power Budgeting

---

Table 19-5: Slot Capability Register Fields and Descriptions (Continued)

Bit(s)	Register Name and Description
5	<b>Hot-Plug Surprise</b> — indicates that it's possible for the user to remove the card from the system without prior notification. This tells the OS to allow for such removal without affecting continued software operation.
6	<b>Hot-Plug Capable</b> — indicates that this slot supports hot plug operation.
14:7	<b>Slot Power Limit Value</b> — specifies the maximum power that can be supplied by this slot. This limit value is multiplied by the scale specified in the next field.
16:15	<b>Slot Power Limit Scale</b> — specifies the scaling factor for the Slot Power Limit Value.
17	<b>ElectroMechanical Interlock Present</b> — indicates that this is implemented for this slot
18	No Command Completed Support— indicates that this slot doesn't generate software notification when a command has been completed. Earlier versions sometimes took a long time to execute hot-plug commands (for example, sometimes taking a second or more to communicate across an I <sup>2</sup> C bus to turn the power on or off), and generated an interrupt when they were finally done. When set this bit means that this Port can accept writes to all fields in the Slot Control register without delay, so there's no need for the notification.
31:19	<b>Physical Slot Number</b> — Indicates the physical slot number associated with this port. It must be hardware initialized to a number that is unique within the chassis. Note that software will need this number to relate the physical slot to the Logical Slot ID (Bus, Device, & Function number for this device).

---

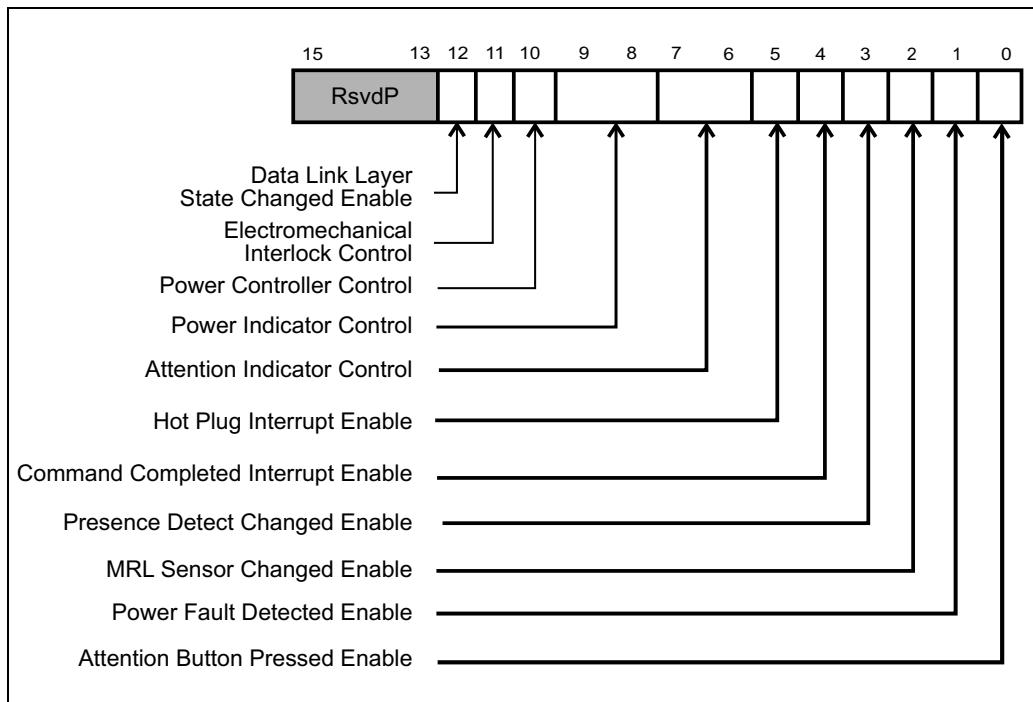
## Slot Power Limit Control

The spec provides a method for software to limit the amount of power consumed by a card installed into an expansion slot or backplane implementation. The registers to support this feature are included in the Slot Capability register.

## Slot Control

Software controls the Hot Plug events through the Slot Control register, shown in Figure 19-6 on page 868. This register permits software to enable various Hot Plug features and control hot plug operations. It's also used to enable interrupt generation as well as enabling the sources of Hot-Plug events that can result in interrupt generation.

Figure 19-6: Slot Control Register



## Chapter 19: Hot Plug and Power Budgeting

---

Table 19-6: Slot Control Register Fields and Descriptions

Bit(s)	Register Name and Description
0	<b>Attention Button Pressed Enable.</b> When set, this bit enables the generation of a hot-plug interrupt (if enabled) or assertion of the Wake# message, when the attention button is pressed.
1	<b>Power Fault Detected Enable.</b> When set, enables generation of a hot-plug interrupt (if enabled) or Wake# message upon detection of a power fault.
2	<b>MRL Sensor Changed Enable.</b> When set, enables generation of a hot-plug interrupt or Wake# (if enabled) message upon detection of a MRL sensor changed event.
3	<b>Presence Detect Changed Enable.</b> When set this bit enables the generation of the hot-plug interrupt or a Wake message when the presence detect changed bit in the Slot Status register is set.
4	<b>Command Completed Interrupt Enable.</b> When set, enables a Hot-Plug interrupt to be generated that informs software that the hot-plug controller is ready to receive the next command.
5	<b>Hot-Plug Interrupt Enable.</b> When set, enables the generation of Hot-Plug interrupts.
7:6	<b>Attention Indicator Control.</b> Writes to the field control the state of the attention indicator and reads return the current state, as follows: <ul style="list-style-type: none"><li>• 00b = Reserved</li><li>• 01b = On</li><li>• 10b = Blink</li><li>• 11b = Off</li></ul>
9:8	<b>Power Indicator Control.</b> Writes to the field control the state of the power indicator and reads return the current state, as follows: <ul style="list-style-type: none"><li>• 00b = Reserved</li><li>• 01b = On</li><li>• 10b = Blink</li><li>• 11b = Off</li></ul>
10	<b>Power Controller Control.</b> Writes to the field switch main power to the slot and reads return the current state: 0b = Power On, 1b = Power Off

# PCI Express Technology

---

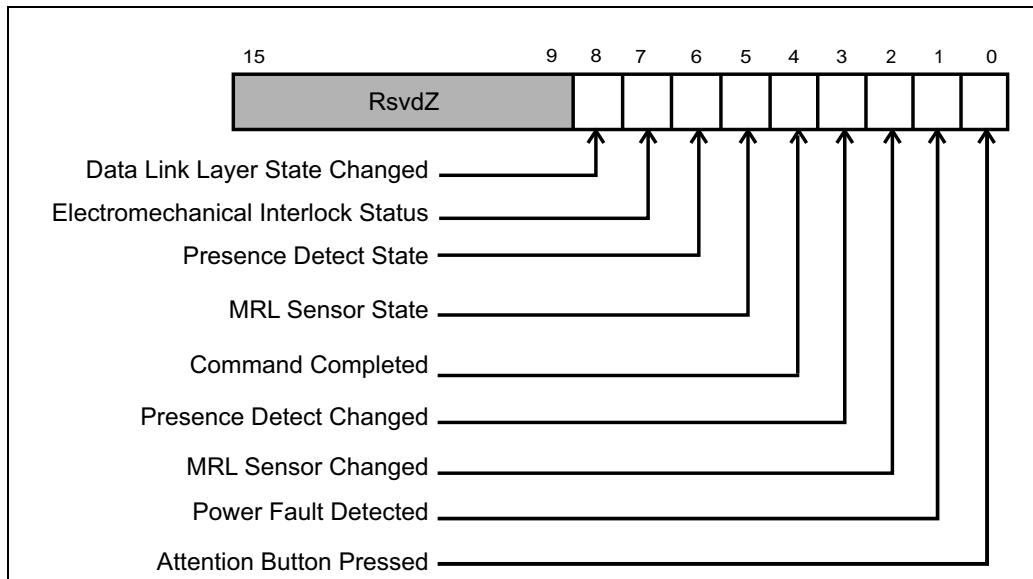
Table 19-6: Slot Control Register Fields and Descriptions (Continued)

Bit(s)	Register Name and Description
11	<b>Electromechanical Interlock Control</b> - If the interlock is implemented, writing a 1b to this bit toggles the state of it while writing a 0b has no effect. Reading this bit always returns a 0b.
12	<b>Data Link Layer State Changed Enable</b> - If the Data Link Layer Link Active Reporting capability is 1b, setting this bit enables software notification when the Data Link Layer Link Active bit changes. If the Data Link Layer Link Active Reporting capability is 0b, then this bit becomes read-only with a value of 0b.

## Slot Status and Events Management

The Hot Plug Controller monitors a variety of events and reports these events to the Hot Plug System Driver. Software can use the “detected” bits to determine which event has occurred, while the status bit identifies that nature of the change. The changed bits must be cleared by software in order to detect a subsequent change. Note that whether these events get reported to the system (via a system interrupt) is determined by the related enable bits in the Slot Control Register.

Figure 19-7: Slot Status Register



## Chapter 19: Hot Plug and Power Budgeting

---

Table 19-7: Slot Status Register Fields and Descriptions

Bit Location	Register Name and Description
0	<b>Attention Button Pressed</b> — If the button is implemented, this bit is set when the Attention Button is pressed.
1	<b>Power Fault Detected</b> — If a Power Controller that supports power fault detection is implemented, this bit is set when it detects a power fault at this slot. The spec notes that it's possible for a power fault to be detected at any time, regardless of the Power Control setting or whether the slot is occupied.
2	<b>MRL Sensor Changed</b> — If an MRL Sensor is implemented, this is set when a MRL Sensor state change is detected. If no sensor is present this bit will always be zero.
3	<b>Presence Detect Changed</b> — set when a change has been detected in the Presence Detect State bit.
4	<b>Command Completed</b> — If the No Command Completed Support bit in the Slot Capabilities register is 0b, then this bit is set when a hot plug command has completed and the Hot Plug Controller is ready to accept another command. Technically, only this last meaning is guaranteed: the controller is ready to accept another command, regardless of whether the previous one has actually completed.
5	<b>MRL Sensor State</b> — when set, indicates the current state of the MRL sensor, if implemented: 0b = MRL Closed, 1b = MRL Open
6	<b>Presence Detect State</b> — this bit indicates the presence of a card in a slot and is required for all Downstream Ports that implement a slot. Its value is the logical “OR” of Physical Layer’s Detection logic and any other side-band detect mechanism implemented for the slot (such as PRSNT1# and PRSNT2#). The big difference between them is that the pins require no power to physically detect the card and can thus report on it without needing the power restored, while using the Physical Layer Detect logic does need power.

Table 19-7: Slot Status Register Fields and Descriptions (Continued)

Bit Location	Register Name and Description
7	<b>Electromechanical Interlock Status</b> —If an Electromechanical Interlock is implemented, this bit indicates whether it is engaged (1b) or disengaged (0b).
8	<b>Data Link State Changed</b> — This bit is set when the Data Link Layer Link Active bit in the Link Status register changes. In response to this event, software must read the Data Link Layer Link Active bit to determine whether the Link is active before sending configuration cycles to the hot plugged device.

---

## Add-in Card Capabilities

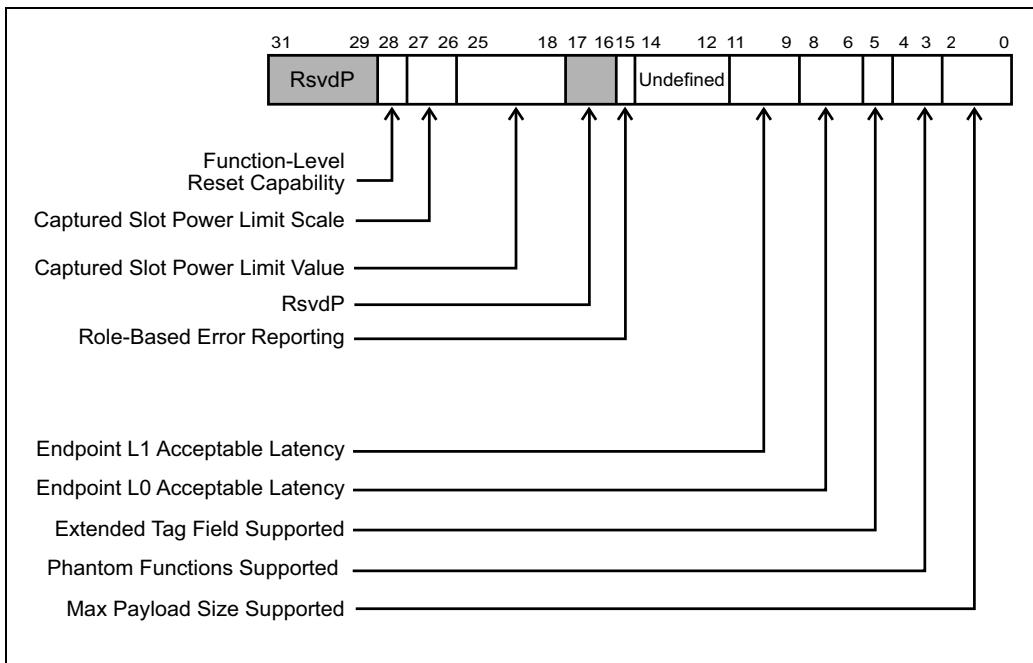
The Device Capability register, seen in Figure 19-8 on page 873, also has fields relevant to add-in cards that record the power reported by the Hot Plug Controller as being available to their slot. This information must be communicated automatically with a Set\_Slot\_Power\_Limit Message whenever either of these takes place:

- A configuration write to the Slot Capabilities register changes the Slot Power Limit Value and Slot Power Limit Scale values.
- The Link transitions from non-DL\_UP to DL\_Up status (unless the Slot Capabilities register has not yet been initialized).

The message updates the Captured Slot Power Limit Value and Scale registers with the values in the message, making this information readily available to its device driver.

## Chapter 19: Hot Plug and Power Budgeting

Figure 19-8: Device Capabilities Register



## Quiescing Card and Driver

### General

Prior to removing a card from the system, two things must occur: the device driver must stop accessing the card, and the card must stop initiating or responding to new Requests. How this is accomplished is OS-specific, but the following must take place:

- The OS must stop issuing new requests to the device's driver or instruct the driver to stop accepting new requests.
- The driver must terminate or complete all outstanding requests.
- The card must be disabled from generating interrupts or Requests.

When the OS commands the driver to quiesce itself and its device, the OS must not expect the device to remain in the system (in other words, it could be removed and not replaced with an identical card).

## Pausing a Driver (Optional)

Optionally, an OS could implement a “Pause” capability to temporarily stop driver activity in the expectation that the same card will be reinserted. If the card is not reinstalled within a reasonable amount of time, however, the driver must be quiesced and then removed from memory.

As an example, the currently-installed card is failing or is being replaced with a later revision as an upgrade. If the operation is to appear seamless from a software and operational perspective, the driver would have to quiesce the device, save the current context (contents of registers, stack and instruction pointer of local micro-controller, etc.) and turn off the power to the slot. The new card could then be installed and powered, and then, when its context is restored, it could resume normal operation where it left off. Of course, if the old card had failed, it may not be possible to simply resume operation.

## Quiescing a Driver That Controls Multiple Devices

If a driver controls multiple cards and it receives a command from the OS to quiesce its activity with respect to a specific card, it must only quiesce its activity with that card and the card itself.

## Quiescing a Failed Card

If a card has failed, it may not be possible for the driver to complete requests previously issued to the card. In this case, the driver must detect the error, terminate the requests without completion, and attempt to reset the card.

---

## The Primitives

This section discusses the hot-plug software elements and the information passed between them. For a review of the software elements and their relationships to each other, refer to Table 19-1 on page 852. Communications between the Hot-Plug Service within the OS and the Hot-Plug System Driver is in the form of requests. The spec doesn’t define the exact format of these requests, but does define the basic request types and their content. Each request type issued to the Hot-Plug System Driver by the Hot-Plug Service is referred to as a *primitive*. They are listed and described in Table 19-8 on page 875.

## Chapter 19: Hot Plug and Power Budgeting

---

*Table 19-8: The Primitives*

Primitive	Parameters	Description
Query Hot-Plug System Driver	<b>Input:</b> None  <b>Return:</b> Set of Logical Slot IDs for slots controlled by this driver.	Requests that the Hot-Plug System Driver return a set of Logical Slot IDs for the slots it controls.
Set Slot Status	<b>Inputs:</b> <ul style="list-style-type: none"> <li>• Logical Slot ID</li> <li>• New slot state (on or off).</li> <li>• New Attention Indicator state.</li> <li>• New Power Indicator state.</li> </ul> <b>Return:</b> Request completion status: <ul style="list-style-type: none"> <li>• status change successful</li> <li>• fault—wrong frequency</li> <li>• fault—insufficient power</li> <li>• fault—insufficient configuration resources</li> <li>• fault—power fail</li> <li>• fault—general failure</li> </ul>	This request is used to control the slots and the Attention Indicator associated with each slot. Good completion of a request is indicated by returning the Status Change Successful parameter. If a fault is incurred during an attempted status change, the Hot-Plug System Driver should return the appropriate fault message (see middle column). Unless otherwise specified, the card should be left in the off state.
Query Slot Status	<b>Input:</b> Logical Slot ID  <b>Return:</b> <ul style="list-style-type: none"> <li>• Slot state (on or off)</li> <li>• Card power requirements.</li> </ul>	This request returns the state of the indicated slot (if a card is present). The Hot-Plug System Driver must return the Slot Power status information.

# PCI Express Technology

---

Table 19-8: The Primitives (Continued)

Primitive	Parameters	Description
Async Notice of Slot Status Change	<b>Input:</b> Logical Slot ID <b>Return:</b> none	This is the only primitive (defined by the spec) that is issued to the Hot-Plug Service by the Hot-Plug System Driver. It is sent when the Driver detects an unsolicited change in the state of a slot. Examples would be a run-time power fault or a card installed in a previously-empty slot with no warning.

---

## Introduction to Power Budgeting

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

The spec states that “power budgeting capability is optional for PCI Express devices implemented in a form factor which does not require hot plug, or that are integrated on the system board.” None of the form factor specs released at the time of this writing required support for hot plug or the power budgeting capability, but these change often.

System power budgeting is always required to support all system board devices and add-in cards. The new capability provides mechanisms for managing the budgeting process for a hot-plug card. Each form factor spec defines the min and max power for a given expansion slot. For example, the CEM spec limits the power an expansion card can consume prior to being fully enabled but, after it is enabled, it can consume the maximum amount of power specified for the slot. In the absence of the power budgeting capability registers, the system designer is responsible for guaranteeing that power has been budgeted correctly and that sufficient cooling is available to support any compliant card installed into the connector.

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

# **Chapter 19: Hot Plug and Power Budgeting**

---

## **The Power Budgeting Elements**

Figure 19-10 illustrates the concept of Power Budgeting for hot plug cards. The role of each element involved in the power budgeting, allocation, and reporting process is listed and described below:

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

---

## **System Firmware**

Written by the platform designers the specific system, this is responsible for reporting system power information. The spec recommends the following power information be reported to the PCI Express power budget manager, which allocates and verifies power consumption and dissipation during runtime:

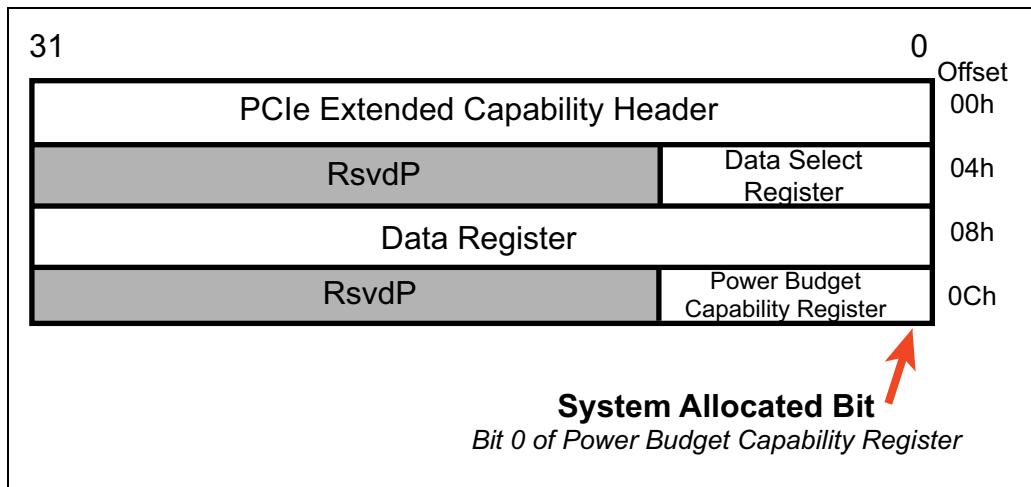
- Total system power available.
- Power allocated to system devices by firmware
- Number and type of slots in the system.

Firmware may also allocate power to PCIe devices that support the power budgeting capability register set, such as a hot-plug device used during boot time. The Power Budgeting Capability register, shown in Figure 19-9 on page 878, contains a System Allocated bit that is hardware initialized (usually by firmware) to notify the power budget manager that power for this device has already been included in the system power allocation. If so, the Power Budget Manager still needs to read and save the power information for the hot-plug devices that were allocated in case they are later removed during runtime.

# PCI Express Technology

---

Figure 19-9: Power Budget Registers



---

## The Power Budget Manager

This initializes when the OS installs and receives power-budget information from system firmware, although the spec does not define the method for delivering this information. This manager is responsible for allocating power for all PCI Express devices including:

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

---

## Expansion Ports

Figure 19-10 on page 880 illustrates a hot plug port that must have the Slot Power Limit and Slot Power Scale fields within the Slot Capabilities register implemented. The firmware or power budget manager must load these fields with a value that represents the maximum amount of power supported by this Port. When software writes to these fields the Port automatically delivers a Set\_Slot\_Power\_Limit message to the device. These fields are also written when software configures a new card that has been added as a hot plug installation.

# **Chapter 19: Hot Plug and Power Budgeting**

---

Spec requirements:

- Any Downstream Port that has a slot attached (the Slot Implemented bit in its PCIe Capabilities register is set) must implement the Slot Capabilities register.
- Software must initialize the Slot Power Limit Value and Scale fields of the Slot Capabilities register of the Downstream Port that is connected to an add-in slot.
- Upstream Ports must implement the Device Capabilities register.
- When a card is installed in a slot and software updates the power limit and scale values in the Downstream Port, that Port will automatically send the Set\_Slot\_Power\_Limit message to the Upstream Port on the installed card.
- The recipient of the Message must use the data payload to limit its power usage for the entire card, unless the card will never exceed the lowest value specified in the corresponding electromechanical spec.

---

## **Add-in Devices**

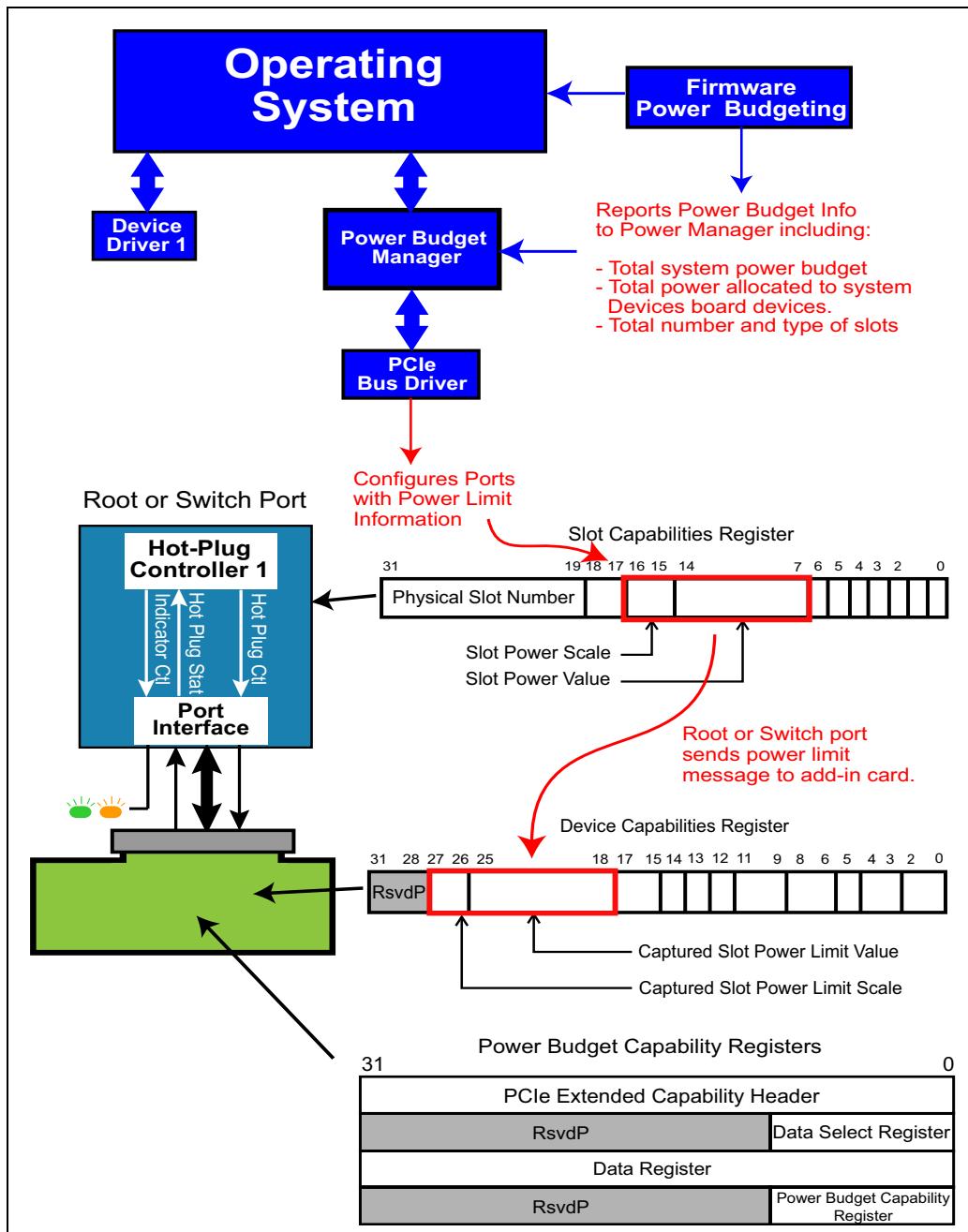
Expansion cards that support the power budgeting capability must include the Slot Power Limit Value and Slot Limit Scale fields within the Device Capabilities register, and the Power Budgeting Capability register set for reporting power-related information.

These devices must not consume more than the lowest power specified by the form factor spec. Once power budgeting software allocates additional power via the Set\_Slot\_Power\_Limit message, the device can consume the power that has been specified, but not until it has been configured and enabled.

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

# PCI Express Technology

Figure 19-10: Elements Involved in Power Budget



# Chapter 19: Hot Plug and Power Budgeting

---

## Slot Power Limit Control

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

## Expansion Port Delivers Slot Power Limit

Software writes to the *Slot Power Limit Value* and *Slot Power Limit Scale* fields of the Slot Capability register to specify the maximum power that can be consumed by the device. Software is required to specify a power value that reflects one of the maximum values defined by the spec. For example, revision 2.0 of the CEM spec defines power usage as listed in Table 19-9.

An interesting note about these values is that a standard-height x1 server card is limited to 10W after a reset and is only allowed to use the full 25W after it's been configured and enabled. Similarly, a x16 graphics card will be limited to 25W until configured and enabled to use the full 75W.

Table 19-9: Maximum Power Consumption for System Board Expansion Slots

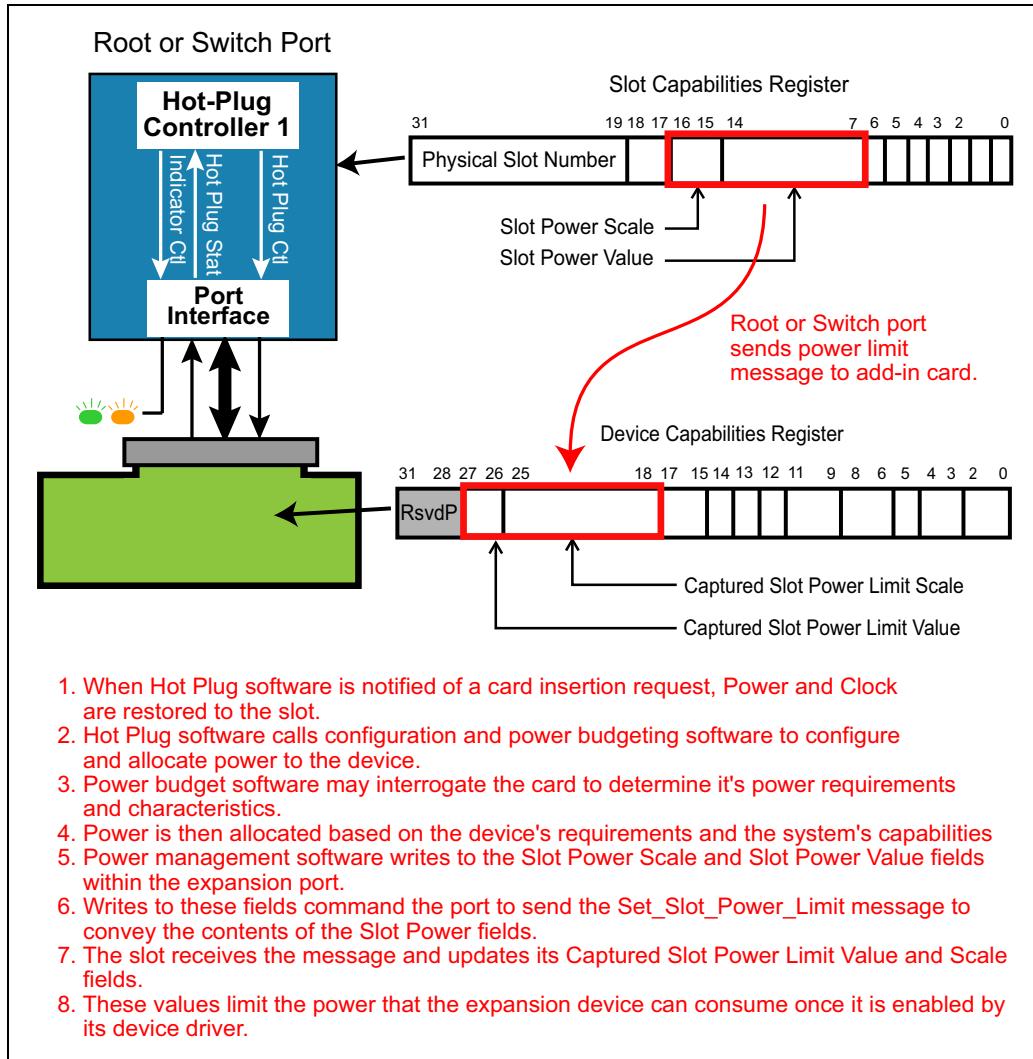
	X1 Link	X4/X8 Link	X16 Link	
Standard Height	10W (max - desktop)	25W (max - server)	25W (max)	25W (max - server) 75W (max - graphics card)
Low Profile Card	10W (max)		25W (max)	25W (max)

In addition to the base CEM spec, two more specs have been defined for higher-powered devices. First is the PCIe x16 Graphics 150W-ATX Spec 1.0, which defines a video card that's able to draw 75W from the card connector and another 75W from a separate 3-pin ATX power connector. The second is the PCIe 225W/300W High Power CEM Spec 1.0, which extends this by adding another 3-pin power connector to achieve 225W, or a 4-pin ATX connector that brings the total to 300W.

# PCI Express Technology

When the Slot Power registers are written by power budget software, the expansion port sends a Set\_Slot\_Power\_Limit message to the expansion device. This procedure is illustrated in Figure 19-11 on page 882.

Figure 19-11: Slot Power Limit Sequence



## Expansion Device Limits Power Consumption

The device driver reads the values from the Captured Slot Power Limit and Scale fields to verify that the power available is sufficient to operate the device. Several conditions may exist:

- Enough power is available to operate the device at full capability. In this case, the driver enables the device by writing to the configuration Command register, permitting the device to consume power up to the limit specified in the Power Limit fields.
- The power available is sufficient to operate the device but not at full capability. In this case, the driver is required to configure the device such that it consumes no more power than specified in the Power Limit fields.
- The power available is insufficient to operate the device. In this case, the driver must not enable the card and must report the inadequate power condition to the upper software layers, which should in turn inform the end user of the problem.
- The power available exceeds the maximum power specified by the form factor spec. This condition should not occur, but, if it does, the device is not permitted to consume power beyond the maximum permitted by the form factor.
- The power available is less than the lowest value specified by the form factor spec. This is a violation of the spec, which states that the expansion port “must not transmit a Set\_Slot\_Power\_Limit Message that indicates a limit lower than the lowest value specified in the electromechanical spec for the slot’s form factor.”

Some expansion devices may consume less power than the lowest limit specified for their form factor. Such devices are permitted to discard the information delivered in the Set\_Slot\_Power\_Limit Messages. When the Slot Power Limit Value and Scale fields are read, these devices return zeros.

---

## The Power Budget Capabilities Register Set

These registers permit power budgeting software to allocate power more effectively based on information provided by the device through its power budget data select and data register. This feature is similar to the data select and data fields within the power management capability registers. However, the power budget registers provide more detailed information to software to aid it in determining the effects of expansion cards that are added during runtime on

# PCI Express Technology

---

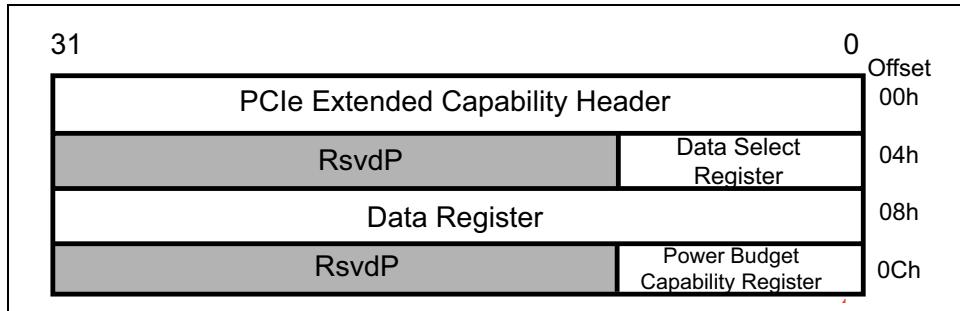
the system power budget and cooling requirements. Through this capability, a device can report the power it consumes:

- from each power rail
- in various power management states
- in different operating conditions

These registers are not required for devices implemented on the system board or on expansion devices that do not support hot plug. Figure 19-12 on page 884 illustrates the power budget capabilities register set and shows the data select and data field that provide the method for accessing the power budget information.

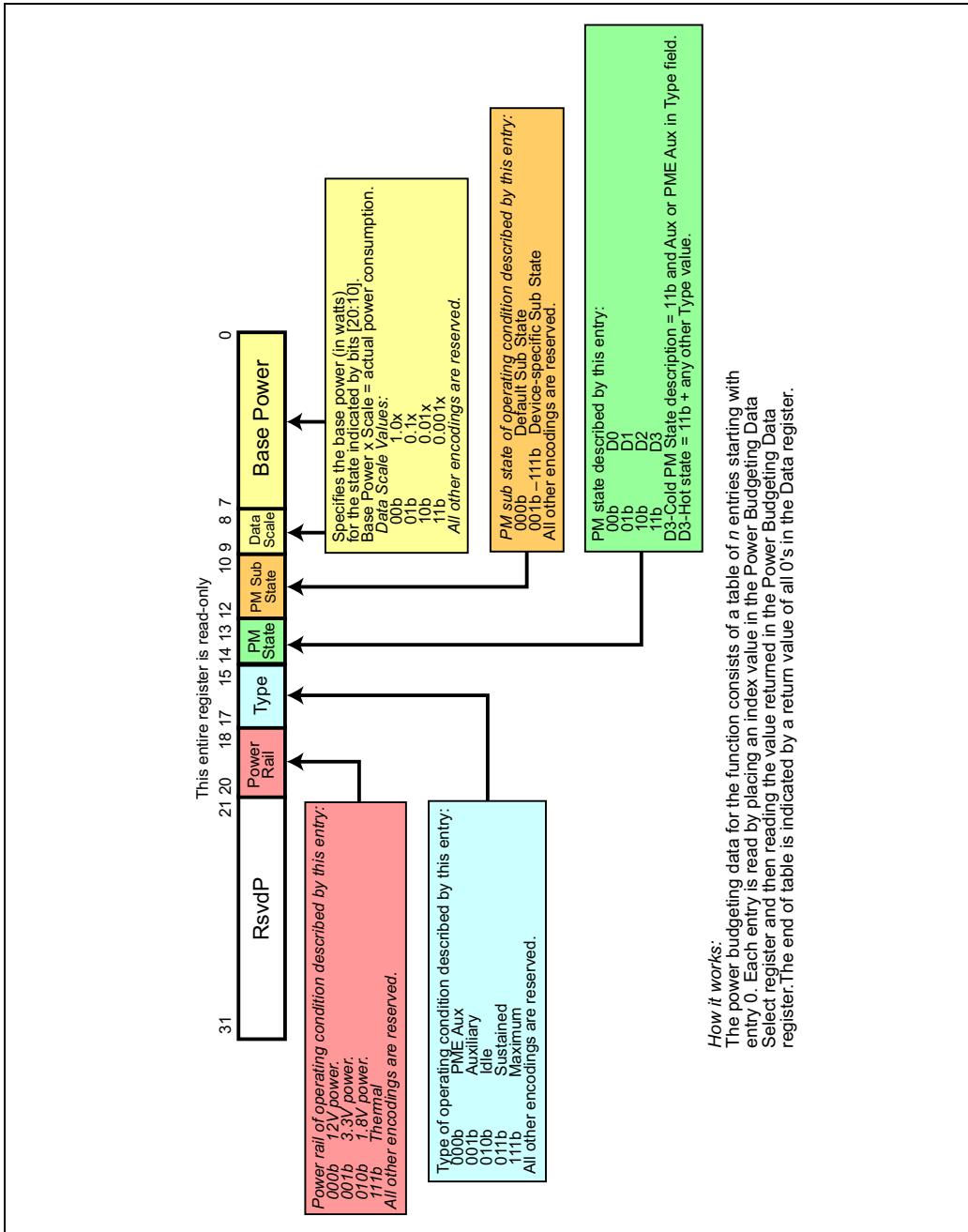
The power budget information is maintained within a table that consists of one or more 32-bit entries. Each table entry contains power budget information for the different operating modes supported by the device. Each table entry is selected via the data select field, and the selected entry is then read from the data field. The index values start at zero and are implemented in sequential order. When a selected index returns all zeros in the data field, the end of the power budget table has been located. Figure 19-13 on page 885 illustrates the format and types of information available from the data field.

*Figure 19-12: Power Budget Capability Registers*



# Chapter 19: Hot Plug and Power Budgeting

Figure 19-13: Power Budget Data Field Format and Definition



## **PCI Express Technology**

---

---

---

# 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.

---

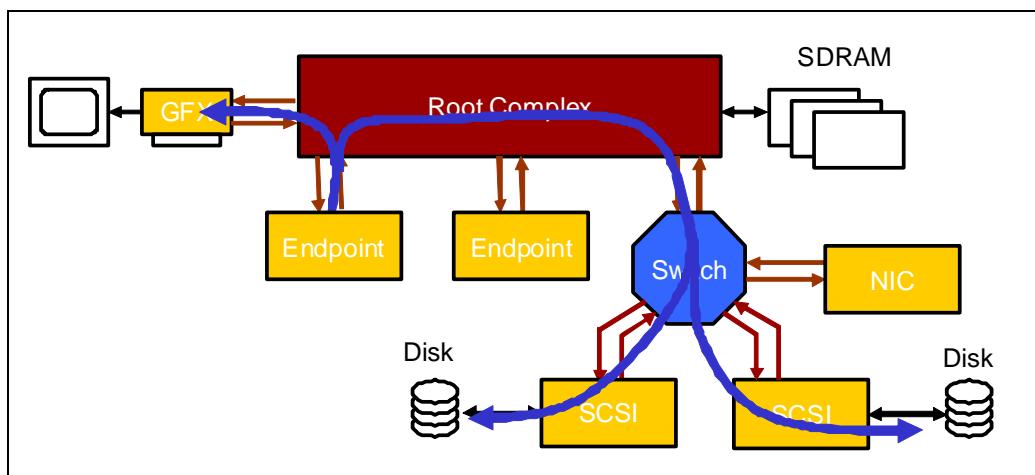
## **Changes for PCIe Spec Rev 2.1**

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



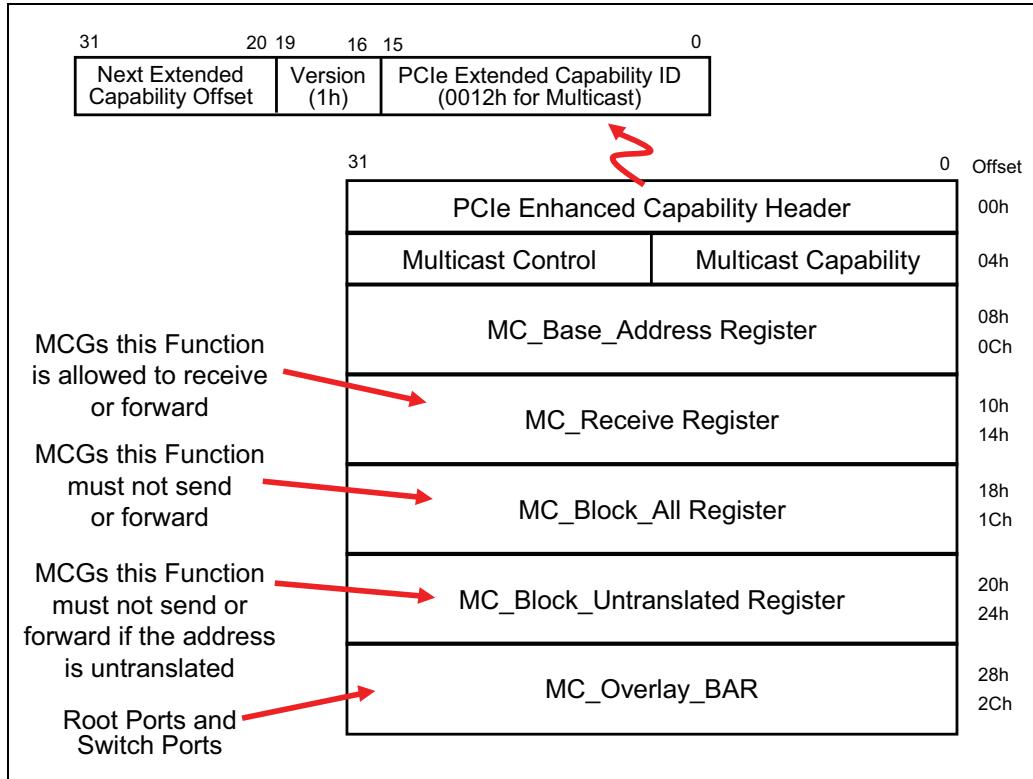
This mechanism is only supported for posted, address-routed Requests, such as Memory Writes, that contain data to be delivered and an address that can be decoded to show which Ports should receive it. Non-posted Requests will not be treated as Multicast even if their addresses fall within the MultiCast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

# Chapter 20: Updates for Spec Revision 2.1

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.

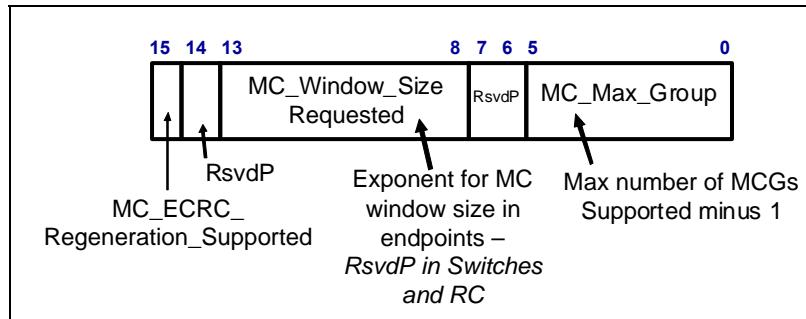
### Multicast Capability

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

# PCI Express Technology

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

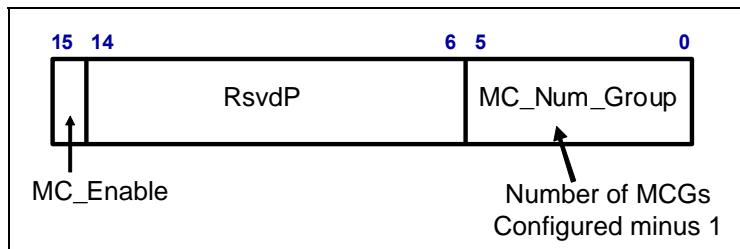


Lastly, bit 15 indicates whether this Function supports regenerating the ECRC value in a TLP if forwarding it involved making address changes to it. Refer to the section called “Overlay Example” on page 895 for more detail on this.

## 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

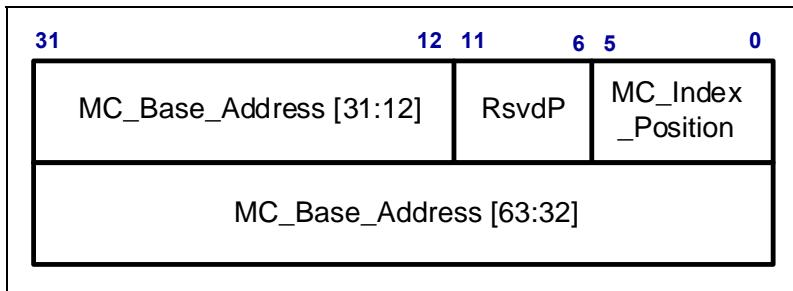


# Chapter 20: Updates for Spec Revision 2.1

## Multicast Base Address

The base address register, shown in Figure 20-5 on page 891, contains the 64-bit starting address of the Multicast Address range for this component. The Multi-Cast Index Position register indicates the bit position within the address where the MultiCast Group (MCG) number is to be found. When the address of an incoming TLP falls within the MultiCast address range starting at this Base Address, the logic will offset into the address itself by the number of bit locations given in the Index Position and interpret the next bits (up to 6 bits, allowing up to 64 groups) as the MCG number for that TLP. The MCG number, in turn, will indicate whether the Port should forward a copy of this TLP.

*Figure 20-5: Multicast Base Address Register*

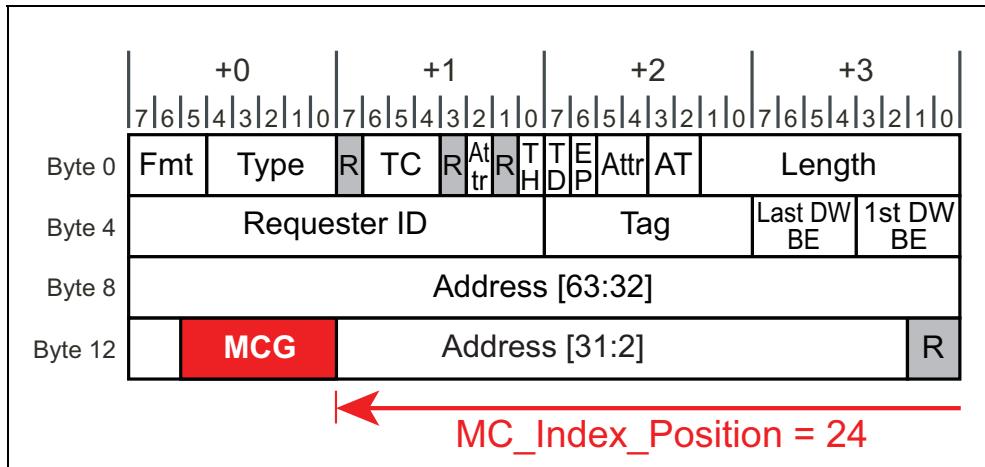


An example of locating the MCG within the address is shown in Figure 20-6 on page 892. Here the Index Position value is 24, so the MCG is found in address bits 25 to 30. Interestingly, since the base address doesn't define the lower 12 bits of the address, the MC Index Position must be 12 or greater to be valid. If it's less than 12 and the MC\_Enable bit is set, the component's behavior will be undefined.

# PCI Express Technology

---

Figure 20-6: Position of Multicast Group Number



## MC Receive

This 64-bit register is a bit vector that indicates for which of the 64 MCGs this Function should accept a copy or this Port should forward a copy. If the MCG value is found to be 47, for example, and bit 47 is set in this register, then this Function should receive it or this Port should forward it.

## MC Block All

This 64-bit register indicates which MCGs an Endpoint Function is blocked from sending and which a Switch or Root Port is blocked from forwarding. This can be programmed in a Switch or Root Port to prevent it from forwarding MultiCast TLPs to an Endpoint that doesn't understand them, for example. A blocked TLP is considered an error condition, and how the error is handled is described in the next section.

## MC Block Untranslated

The meaning and use of this 64-bit register is almost identical to the Block All register except that it doesn't apply to TLPs whose AT header field shows them to be translated. This mechanism can be used to set up a Multicast window that is protected in that it can only receive translated addresses.

If a TLP is blocked because of the setting of either of these two blocking registers, it's handled as an MC Blocked TLP, meaning it gets dropped and the Port

## Chapter 20: Updates for Spec Revision 2.1

---

or Function logs and signals this as an error. Logging the error involves setting the Signaled Target Abort bit in its Status register or its Secondary Status register, as appropriate. That's barely enough information to be useful, though, so the spec highly recommends that Advanced Error Reporting (AER) registers be implemented in Functions with Multicast capability to facilitate isolating and diagnosing faults.

The spec notes that this register is required in all Functions that implement the MC Capability registers, but if an Endpoint Function doesn't implement the ATS (Address Translation Services) registers, the designer may choose to make these bits reserved.

---

### Multicast Example

At this point, an example will help to illustrate how these registers can be used to set up a multicast environment. To set this up, let's first give the relevant registers some values:

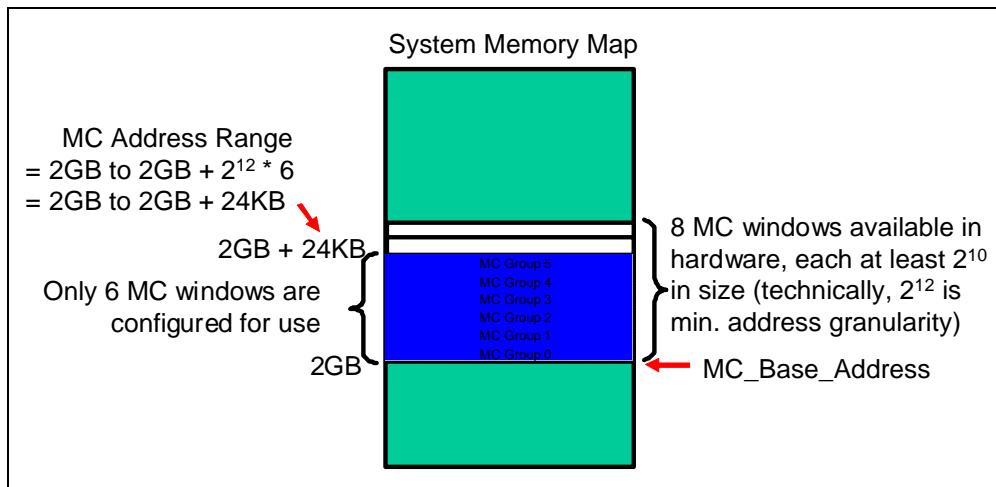
- MC\_Base\_Address = 2GB (Starting address for the multicast range)
- MC\_Max\_Group = 7 (Meaning 8 windows are possible for this design)
- MC\_Window\_Size\_Requested = 10 (Meaning  $2^{10}$  or 1KB size was requested by an Endpoint)
- MC\_Index\_Position = 12 (Meaning the actual size of each window is  $2^{12}$ )
- MC\_Num\_Group = 5 (Meaning software only configured 6 of the available multicast windows).

Based on those register settings, the image in Figure 20-7 on page 894 illustrates the result. The multicast window range is shown starting at 2GB and ranging as high as  $2GB + 8 * (\text{the window size})$ . However, only 6 are enabled by software, so the actual multicast address range is from 2GB to  $2GB + 24KB$ . The windows are all the same size and correspond to the MCGs: MCG 0 is the first window, 1 is the next window, and so on.

# PCI Express Technology

---

Figure 20-7: Multicast Address Example



---

## MC Overlay BAR

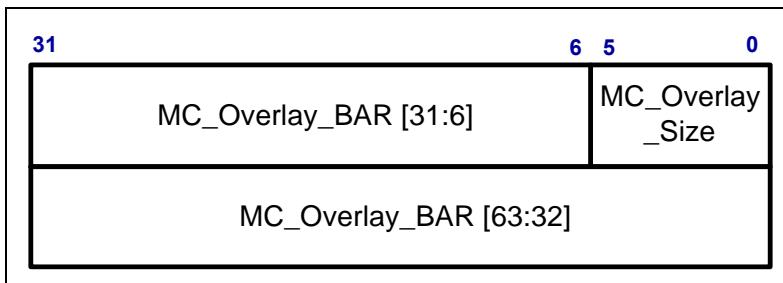
This last set of registers are required for Switch and Root Ports that implement Multicasting, but they're not implemented in Endpoints. The motivation for this BAR is that it allows two special cases. First, a Port can forward TLPs downstream if they hit in a multicast window even if the Endpoint wasn't designed for multicasting. Second, a Port can forward multicast TLPs upstream to system memory. In both cases, this is accomplished by replacing part of the Request's address with an address that will be recognized by the target. Doing so allows a single BAR in a component to serve as a target for both unicast and multicast writes even if it wasn't designed with multicast capability.

As shown in Figure 20-8 on page 895, this register block consists of an address that will be overlaid onto the outgoing TLP, and a 6-bit Overlay Size indicator. The size referred to here is simply the number of bits from the original 64-bit address that will be retained, while all the others will be replaced by the Overlay BAR bits. The spec mistakenly refers to this in at least one place as the size in bytes, but in other places it's made clear that it is a bit number. Note that the overlay size value must be 6 or higher to enable the overlay operation. If the size is given as 5 or lower, no overlay will take place and the address is unchanged.

## Chapter 20: Updates for Spec Revision 2.1

---

Figure 20-8: Multicast Overlay BAR



---

### Overlay Example

Now consider the case in which an address overlay is desired, as shown in Figure 20-9 on page 896. Here the address of a TLP to be forwarded, ABCD\_BEEFh, falls within the defined multicast range (also referred to as a multicast hit) and the egress Port has been configured with valid values in the Overlay BAR.

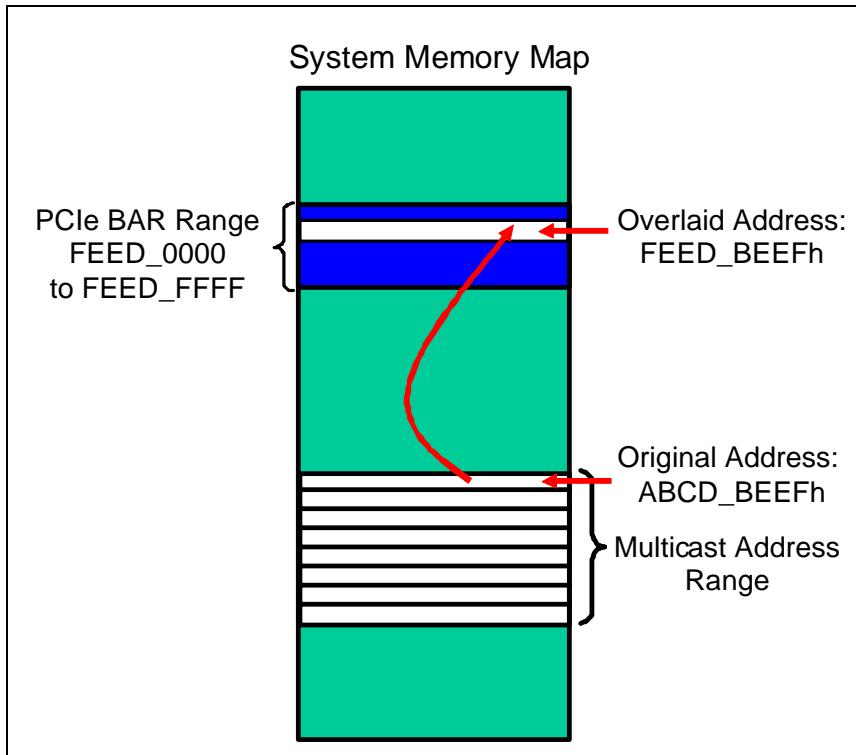
The overlay case creates the unusual situation with the ECRC value that was mentioned earlier in the description of the Multicast Capability register. If the TLP whose address is being changed by the overlay includes an ECRC, that value would be rendered incorrect by this change. Switches and Root Ports optional support regenerating the ECRC based on the new address so that it still serves its purpose going forward. If the routing agent does not support it, the ECRC is simply dropped and the TD header bit is forced to zero to avoid any confusion.

A potential problem can arise with ECRC regeneration. If the incoming TLP already had an error but the ECRC value is regenerated because the address was modified, that would inadvertently hide the original error. To avoid that, the routing agent must verify the original ECRC first. If it finds an error, it must force a bad ECRC on the outgoing TLP by inverting the calculated ECRC value before appending it to ensure that the target will see it as an error condition.

# PCI Express Technology

---

Figure 20-9: Overlay Example



---

## Routing Multicast TLPs

When a Switch or Root Port detects an MC hit (address falls within the MC range) normal routing is suspended. The MCG is extracted from the address and is compared to the MC\_Receive register of all the Ports to see which of them should forward a copy of this TLP. Ports whose corresponding Receive register bit is set will forward a copy of the TLP unless their corresponding MC Blocked register bit is also set. If no Ports forward the TLP and no Functions consume it, it is silently dropped. To prevent loops, a TLP is never forwarded back out on its ingress Port, with the possible exception of an ACS case.

Endpoints extract the MCG and compare it with their Receive register. If there's no match, the TLP is silently dropped. If the Endpoint doesn't support Multicasting, it will treat the TLP as having an ordinary address.

## Congestion Avoidance

The use of Multicasting will increase the amount of system traffic in proportion to the percentage of MC traffic, which leads to the risk of packet congestion. To avoid creating backpressure, MC targets should be designed to accept MC traffic “at speed”, meaning with minimal delay. To avoid oversubscribing the Links, MC initiators should limit their packet injection rate. A system designer would be wise to choose components carefully to handle this. For example, using Switches and Root Ports whose buffers are big enough to handle the expected traffic, and Endpoints that are able to accept their incoming MC packets quickly enough to avoid trouble.

---

## Performance Improvements

System performance is enhanced with the addition of four new features:

1. AtomicOps to replace the legacy transaction locking mechanism
2. TLP Processing Hints to allow software to suggest caching options
3. ID-Based Ordering to avoid unnecessary latency
4. Alternative Routing-ID Interpretation to increase the number of Functions available in a device.

---

## AtomicOps

Processors that share resources or otherwise communicate with each other sometimes need uninterrupted, or “atomic”, access to system resources to do things like testing and setting semaphores. On parallel processor buses this was accomplished by locking the bus with the assertion of a Lock pin until the originator completed the whole sequence (a read followed by a write), during which time other processors were not allowed to initiate transactions on the bus. PCI included a Locked pin to apply this same model on the PCI bus as on the processor bus, allowing this protocol to be used with peripheral devices.

This model worked but was slow on the shared processor bus and even worse when going onto the PCI bus. That’s one reason why PCIe limited its use only to Legacy devices. However, the increasing use of shared processing in today’s PCs, such as graphics co-processors and compute accelerators, has brought this issue back to the fore because the different compute engines need to be able to share an atomic protocol. The way this problem was resolved on PCIe was to introduce three new commands that can each do a series of things atomically

# PCI Express Technology

---

within the target device rather than requiring a series of separate uninterrupted commands on the interface. These new commands, called AtomicOps, are:

1. FetchAdd (Fetch and Add) - This Request contains an “add” value. It reads the target location, adds the “add” value to it, stores the result in the target location and returns the original value of the target location. This could be used in support of atomically updating statistics counters.
2. Swap (Unconditional Swap) - This Request contains a “swap” value. It reads the target location, writes the “swap” value into it, and returns the original target value. This could be useful for atomically reading and clearing counters.
3. CAS (Compare and Swap) - This Request contains both a “compare” value and a “swap” value. It reads the target location, compares it against the “compare” value and, if they’re equal, writes in the “swap” value. Finally, it returns the original value of the target location. This can be useful as a “test and set” mechanism for managing semaphores.

Both Endpoints and Root Ports are optionally allowed to act as AtomicOp Requesters and Completers, which might seem unexpected because, in PCs at least, this kind of transaction is usually only initiated by the central processor. But modern systems can include an Endpoint acting as a co-processor, in which case it would need to be able to use AtomicOps to properly handle the protocol. All three commands support 32-bit and 64-bit operands, while CAS also supports 128-bit operands. The actual size in use will be given in the Length field in the header. Routing elements like Switch Ports and Root Ports with peer-to-peer access will need to support the AtomicOp routing capability to be able to recognize and route these Requests.

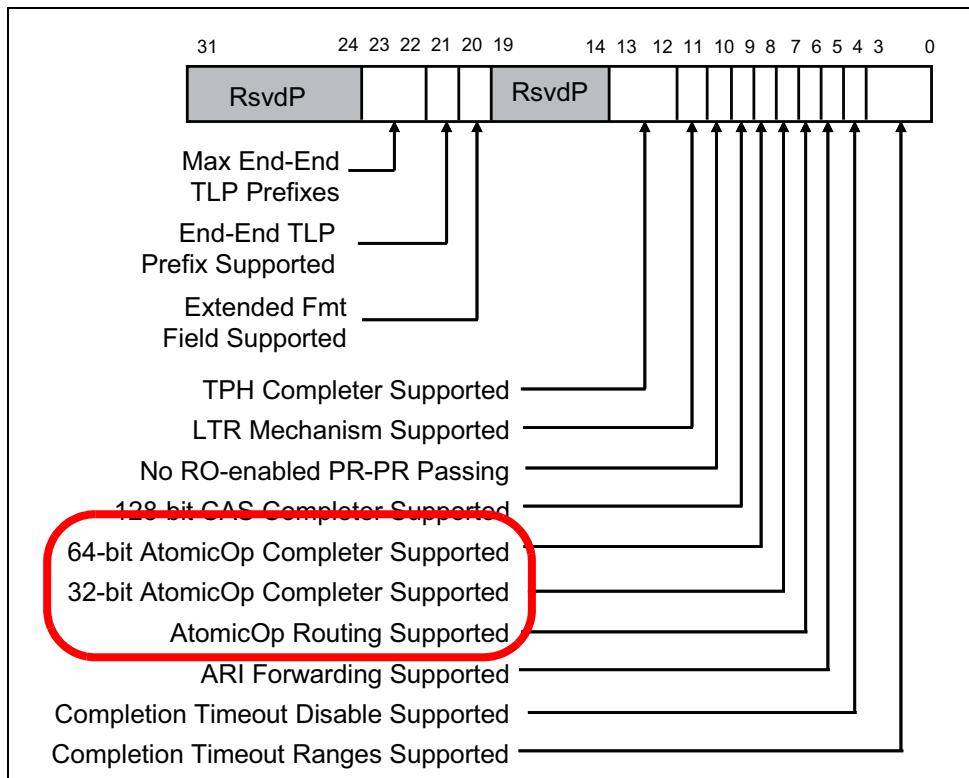
A question naturally arises as to how the system (Root Complex) will be instructed to generate these new commands in response to processor activity, since there may not be a directly-analogous processor bus command. The spec suggests two approaches. First, the Root could be designed to recognize specific processor activity and interpret that to “export” a PCIe AtomicOp in response. Second, a register-based approach similar to the one used for legacy Configuration access could be used. In that case, one register might give the target address while another specified which command should be generated and the combination of the two would generate the Request.

AtomicOp Completers can be identified by the presence of the three new bits in the Device Capabilities 2 register, as shown in Figure 20-10 on page 899. Bit 6 of this register also identifies whether routing elements are capable of routing AtomicOps.

## Chapter 20: Updates for Spec Revision 2.1

Legacy PCI does not comprehend AtomicOps, of course, and there is no straight-forward way to translate them into PCI commands. For that reason, PCIe-to-PCI bridges do not support AtomicOps. If atomic access is needed on that bus it would have to be done with the legacy locked protocol and the spec states that Locked Transactions and AtomicOps can operate concurrently on the same platform.

Figure 20-10: Device Capabilities 2 Register



### TPH (TLP Processing Hints)

Adding hints about how the system should handle TLPs targeting memory space can improve latency and traffic congestion. The spec describes this special handling basically as providing information about which of several possible cache locations in the system would be the optimal place for a temporary copy

# PCI Express Technology

---

of a TLP. The spec makes note of the fact that, since the usage described for TPH relates to caching, it wouldn't usually make sense to use them with TLPs targeting Non-prefetchable Memory Space. If such usage was needed, it would be essential to somehow guarantee that caching such TLPs did not cause undesirable side effects.

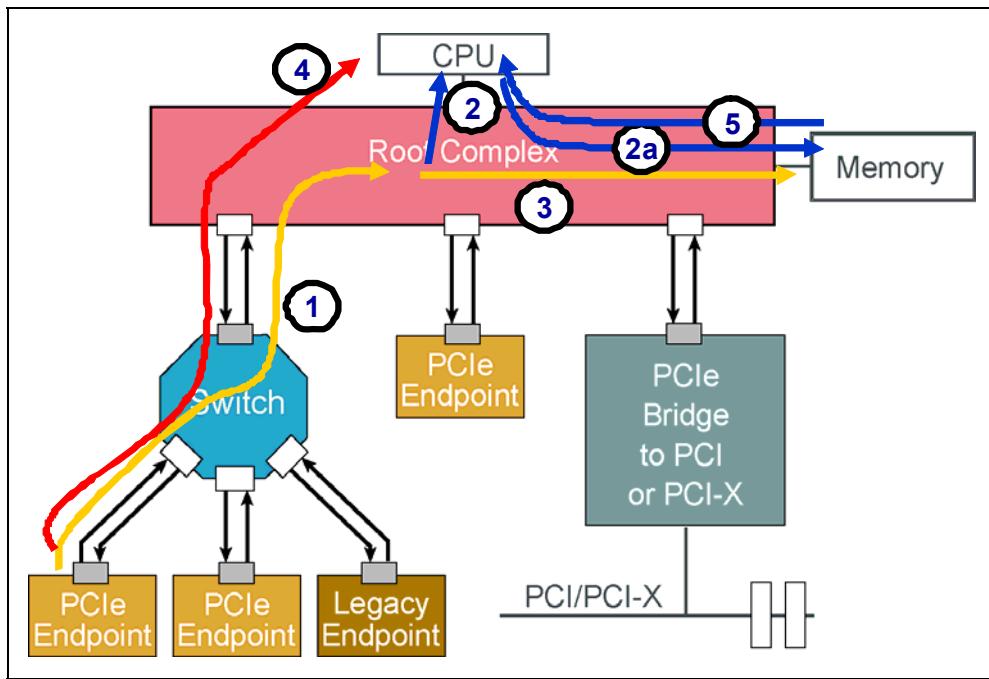
## TPH Examples

**Device Write to Host Read.** To help clarify the motivation for TPH, consider the example shown in Figure 20-11 on page 901. Here the Endpoint is writing data into memory for later use by the CPU. The sequence is as follows:

1. First, the Endpoint sends a memory write TLP containing an address that maps to the system memory. The packet gets routed to the Root Complex (RC).
2. The RC recognizes this as an access to a cacheable memory space and pauses its progress while it snoops the CPU cache. This may result in a write-back cycle from the CPU to update the system memory before the transaction can proceed, and this is shown as step 2a.
3. Once any write backs have finished, the RC allows the write to update the system memory.
4. At some point, the Endpoint notifies the CPU about data delivery.
5. Finally, the CPU fetches the data from memory to complete the sequence.

## Chapter 20: Updates for Spec Revision 2.1

Figure 20-11: TPH Example



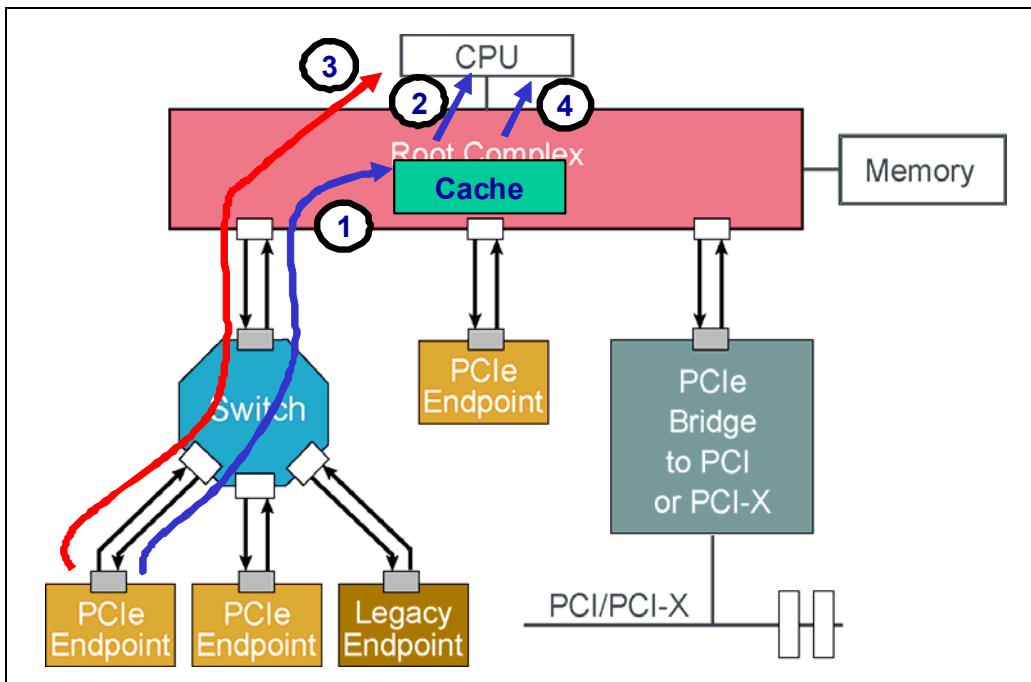
This sequence works but there's an opportunity for performance improvement by adding an intermediate cache in the system. To illustrate this, consider the example shown in Figure 20-12 on page 902. From the perspective of the Endpoint, the operation is the same but the knows to handle it differently. The steps now are as follows:

1. The Endpoint does the same memory write but this time TPH bits are included. The write is forwarded to the RC by the Switch as before.
2. The RC understands that this memory access must be snooped to the CPU as before. However, once the snoop has been handled, the RC is informed by the TPH bits to store this TLP in an intermediate cache rather than going to system memory.
3. The Endpoint notifies the CPU that the data item has been delivered.
4. The CPU reads from the specified address, but now the data is found in the intermediate cache and so the request does not go to system memory. This has the usual benefits we'd expect from a cache design: faster access time as well as reduced traffic for the system memory.

# PCI Express Technology

This is a simple Device Write to Host Read (DWHR) example to illustrate the concept but it wouldn't be hard to imagine a more complex system with a much larger topology in which there could be other caches placed in Switches or other locations to achieve the same benefits for other targets.

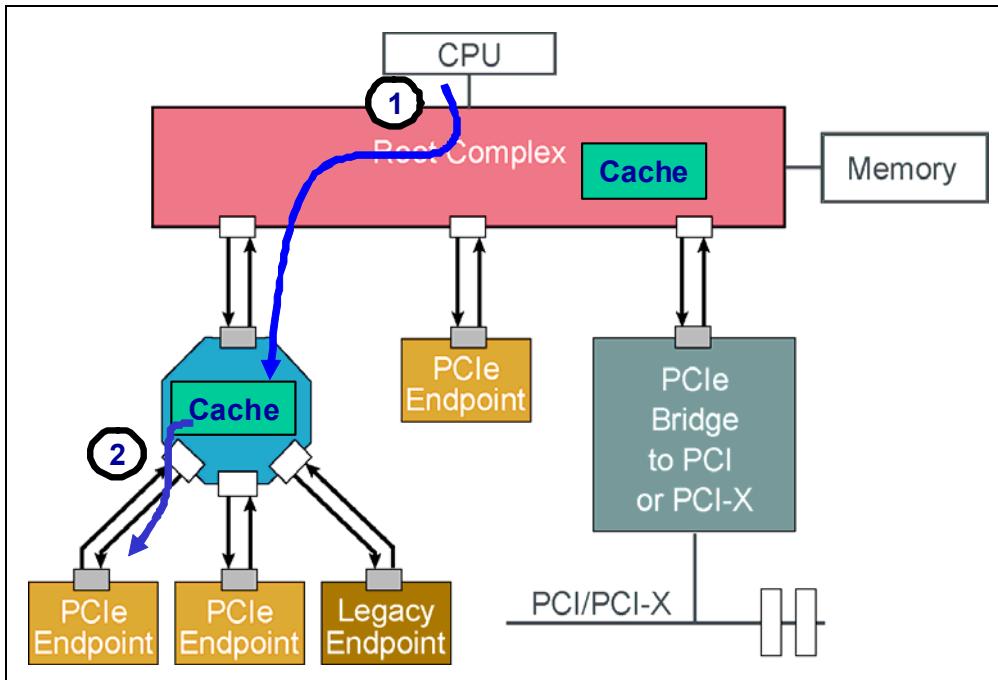
Figure 20-12: TPH Example with System Cache



**Host Write to Device Read.** To illustrate the concept going the other way (called Host Write to Device Read or HWDR), consider the example shown in Figure 20-13 on page 903. In this example, the CPU initiates a memory write whose address targets the PCIe Endpoint in step one. The packet contains TPH bits that tell the RC that it should be stored in an intermediate cache near the target, instead of the cache in the RC that was used in the previous example. In this case a cache built into the Switch serves the purpose. The TLP is then forwarded on to the target Endpoint in step two. This model is beneficial when the data is updated infrequently but read often by the Endpoint. That allows several memory reads that would normally go to system memory to be handled by the cache instead, off loading both the Link from the Switch to the RC and the path to memory.

## Chapter 20: Updates for Spec Revision 2.1

Figure 20-13: TPH Usage for TLPs to Endpoint

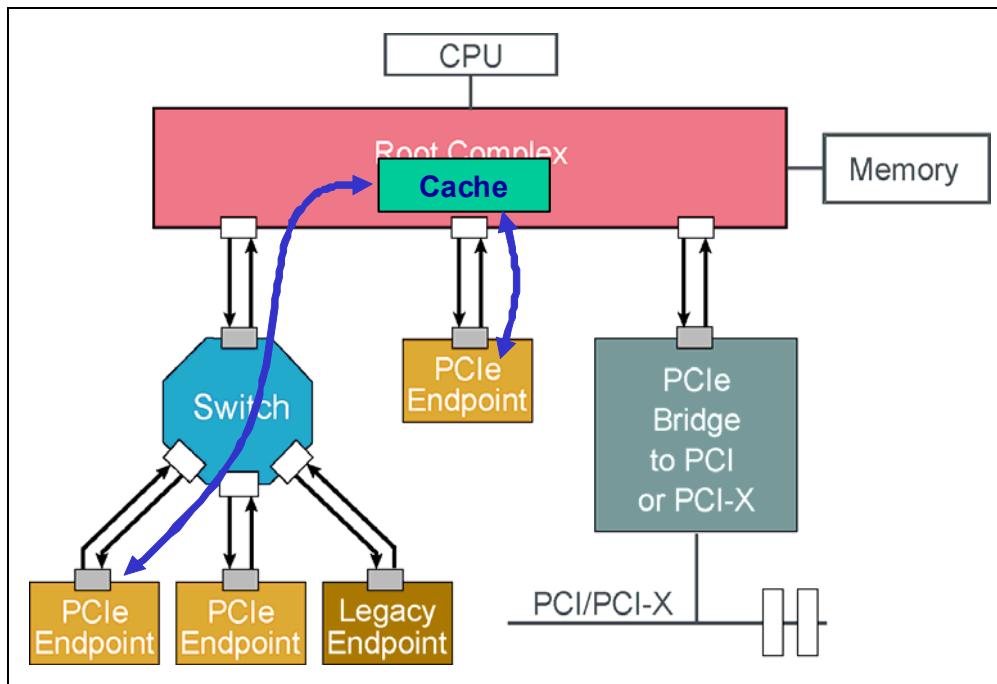


**Device to Device.** One last example is illustrated in Figure 20-14 on page 904, where two Endpoints communicate with each other (called Device Read/Write to Device Read/Write or D\*D\*) through a shared memory location that is directed by TPH bits to an intermediate cache. In this case, both may update different locations that they need to handle as “read mostly”, or one Endpoint may update data that the other needs to read several times. In both cases, using the intermediate cache improves system performance.

# PCI Express Technology

---

Figure 20-14: TPH Usage Between Endpoints

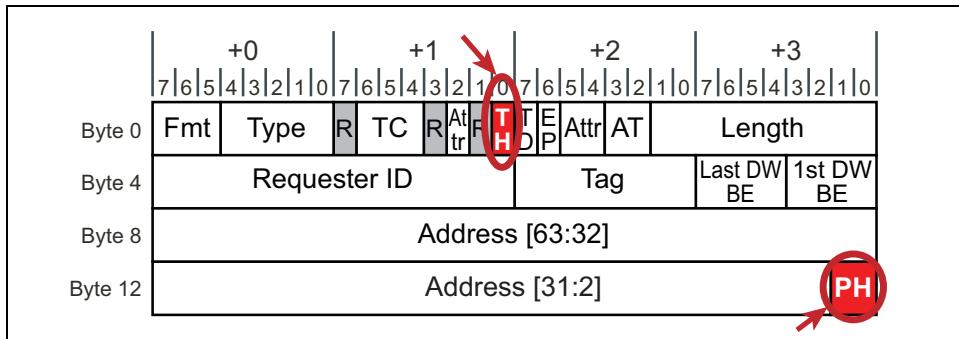


## TPH Header Bits

Several bits in the TLP header describe how the hints are used. First, as shown in the middle at the top of Figure 20-15 on page 905, the TH (TLP Hints) bit reports whether the optional TPH bits are in use for the TLP. When set, the PH (Processing Hint bits) indicate the next level of information.

## Chapter 20: Updates for Spec Revision 2.1

Figure 20-15: TPH Header Bits



When the TH bit is set the PH bits, shown at the bottom right of Figure 20-15 on page 905, take the place of what were the two reserved LSBs in the address field. For a 32-bit address, these are byte 11 [1:0], while for the 64-bit address shown, they are byte 15 [1:0]. Their encoding is described in Table 20-1 on page 905. These hints are provided by the Requester based on knowledge of the data patterns in use, which is information that would be difficult for a Completer to deduce on its own.

Table 20-1: PH Encoding Table

PH [1:0]	Processing Hint	Usage Model
00b	Bi-directional data structure	Indicates frequent read/write access by Host and device.
01b	Requester	D*D* (device-to-device transfers). Indicates frequent read/write access by device. The asterisk means either device could be reading or writing.
10b	Target	DWHR, HWDR (device-to-host or host-to-device transfers). Indicates frequent read/write access by Host.
11b	Target with Priority	Same as Target but with additional temporal re-use priority information. Indicates frequent read/write access by Host and high temporal locality for accessed data.

# PCI Express Technology

---

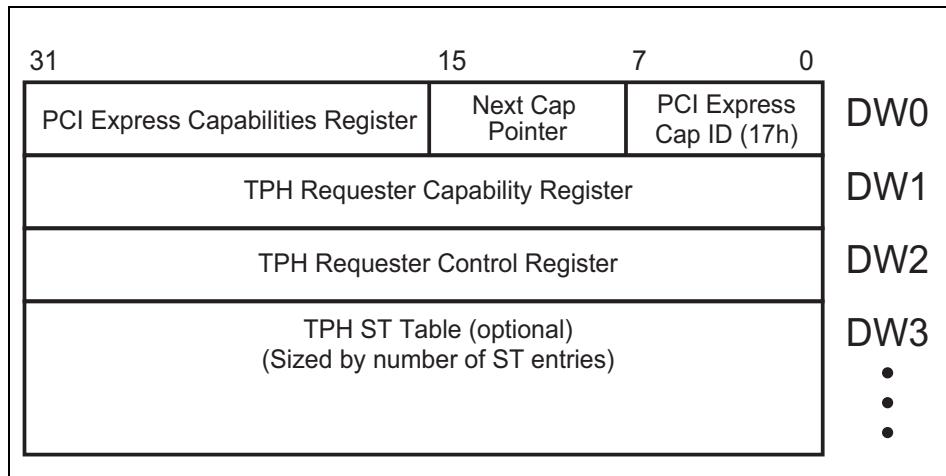
The next level of information is the Steering Tag byte that provides system-specific information regarding the best place to cache this TLP. Interestingly, the location of this byte in the header varies depending on the Request type. For Posted Memory Writes the Tag field is repurposed to be the Steering Tag (no completion will be returned so the Tag isn't needed), while for Memory Reads the two Byte Enable fields are repurposed for it (byte enables are not needed for pre-fetchable reads). The meaning of the bits is implementation specific but they need to uniquely identify the location of the desired cache in the system.

Two formats for TPH are described in the spec and this level of hint information (TH + PH + 8-bit Steering Tag), called Baseline TPH, is the first and is required of all Requests that provide TPH. The second format uses TLP Prefixes to extend the Steering Tags (see "TLP Prefixes" on page 908 for more detail).

## Steering Tags

These values are programmed by software into a table to be used during normal operation. The spec recommends that the table be located in the TPH Requester Capability structure, shown in Figure 20-16 on page 906, but it can alternatively be built into the MSI-X table instead. Only one or the other of these table locations can be used for a given Function. The location is given in the ST Table Location field [10:9] of the Requester Capability register, shown in Figure 20-17 on page 907. The encoding of these 2 bits is shown in Table 20-2 on page 907.

Figure 20-16: TPH Requester Capability Structure



## Chapter 20: Updates for Spec Revision 2.1

Figure 20-17: TPH Capability and Control Registers

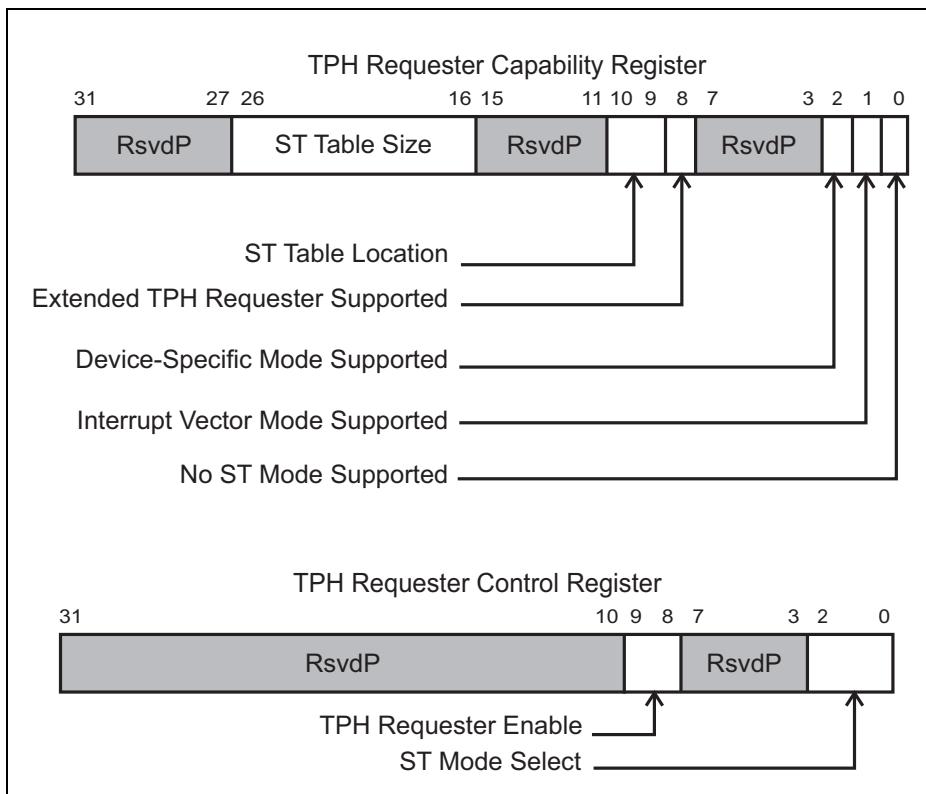


Table 20-2: ST Table Location Encoding

Bits [10:9]	ST Table Location
00b	Not present
01b	Located in the Requester Capability structure
10b	Located in the MSI-X table
11b	Reserved

# PCI Express Technology

---

The Requester Capability register lists the number of entries in the ST Table in bits [26:16]. Each table entry is 2 bytes wide, and the ST Table implemented in the TPH Capability register set is shown in Figure 20-18 on page 908, where entry zero is highlighted. The Requester Capability register also describes which ST Modes are supported for the Requester with the 3 LSBs:

- **No ST** - uses zeros for ST bits. Selected in the TPH Requester Control register's ST Mode Select field when the value = 000b.
- **Interrupt Vector** - uses the interrupt vector number as the offset into the table, meaning the values are contained in the MSI-X table. (ST Mode Select value = 001b.)
- **Device-Specific** - uses a device-specific method to offset into the ST Table in the TPH Capability structure because the ST values are located there. This is the recommended implementation, although how a given Request is associated with a particular ST entry is outside the scope of the spec. (ST Mode Select value = 010b.)
- All other ST Mode Select encodings are reserved for future use.

Figure 20-18: TPH Capability ST Table

31	24	23	16	15	8	7	0
ST Upper Entry (1)	ST Lower Entry (1)		ST Upper Entry (0)	ST Lower Entry (0)			
ST Upper Entry (3)	ST Lower Entry (3)		ST Upper Entry (2)	ST Lower Entry (2)			
●			●				
●			●				
ST Upper Entry (Table Size)	ST Lower Entry (Table Size)		ST Upper Entry (Table Size - 1)	ST Lower Entry (Table Size - 1)			

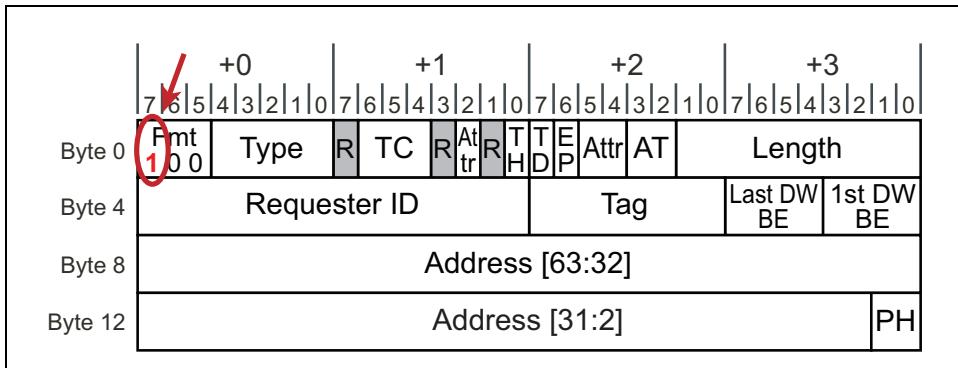
## TLP Prefixes

The Steering Tag bits can be extended with the addition of optional TLP Prefixes if needed. When one or more Prefixes are given with the TLP, the header reports it by setting the most significant bit in the Format field, as shown in Figure 20-19 on page 909.

# Chapter 20: Updates for Spec Revision 2.1

---

Figure 20-19: TPH Prefix Indication



---

## IDO (ID-based Ordering)

Transaction ordering rules are important for proper traffic flow, but there are times when it's not necessary and latencies can be improved in those cases. In particular, TLPs from different Requesters are very unlikely to have dependencies between them, so this feature allows software to enable them to be re-ordered for improved performance. The details of this operation are described in the section called "ID Based Ordering (IDO)" on page 301.

---

## ARI (Alternative Routing-ID Interpretation)

The motivation for this optional feature is to increase the number of Function numbers available to Endpoints. Device numbers were useful in a shared-bus architecture like PCI but are not usually needed in a point-to-point architecture. Consequently, the spec writers chose to allow devices to interpret the destination for ID-routed commands differently. This was accomplished by defining the Device number to always be zero and then allowing the Function number to use the 5 bits in the ID that were previously the Device number. Effectively, the Device number goes away while the Function number grows to 8 bits. The target for a TLP that uses ARI will need to be enabled to recognize it before software can use this feature, but Routing elements in the path to it don't have to be aware of this. They're only looking at the bus number to determine the routing.

## Power Management Improvements

There are four additions that improve the system's ability to manage power effectively, and they are listed here. All of these are covered in Chapter 16, entitled "Power Management," on page 703.

---

### DPA (Dynamic Power Allocation)

A new set of extended configuration registers defines up to 32 sub-states below D0. This allows software to easily make changes to a device's power state without incurring the latency penalty of going all the way to the D1 device power state. To learn more on this, see "Dynamic Power Allocation (DPA)" on page 714

---

### LTR (Latency Tolerance Reporting)

Allowing Endpoints to report the latencies they can tolerate in response to their requests enables system software to make better choices regarding system response time and sleep states. To learn more about this, see "LTR (Latency Tolerance Reporting)" on page 784.

---

### OBFF (Optimized Buffer Flush and Fill)

Similarly, allowing the system to report the preferred time slots during which Endpoints should or should not initiate DMA or interrupt traffic helps coordinate system sleep times and improve power management. For more on this, see "OBFF (Optimized Buffer Flush and Fill)" on page 776.

---

### ASPM Options

This change simply permits devices to support no ASPM Link power management if they choose to do so. In the previous spec versions, support for L0s was mandatory, but now it becomes optional.

## Configuration Improvements

A few configuration registers were added to improve software visibility and control of devices.

---

### Internal Error Reporting

This is intended to provide a standardized way of reporting internal problems for devices like switches that don't have a driver to handle that for them. It also adds the capability to track multiple TLP headers when they result in errors instead of just one as before. This topic is covered in the section on errors called "Internal Errors" on page 667.

---

### Resizable BARs

This new set of extended configuration registers allows devices that use a large amount of local memory to report whether they can work with smaller amounts and, if so, what sizes are acceptable. Software that knows to look for them can find the new registers, shown in Figure 20-20 on page 912, and program them to give the appropriate memory size for the platform based on the competing requirements of system memory and other devices.

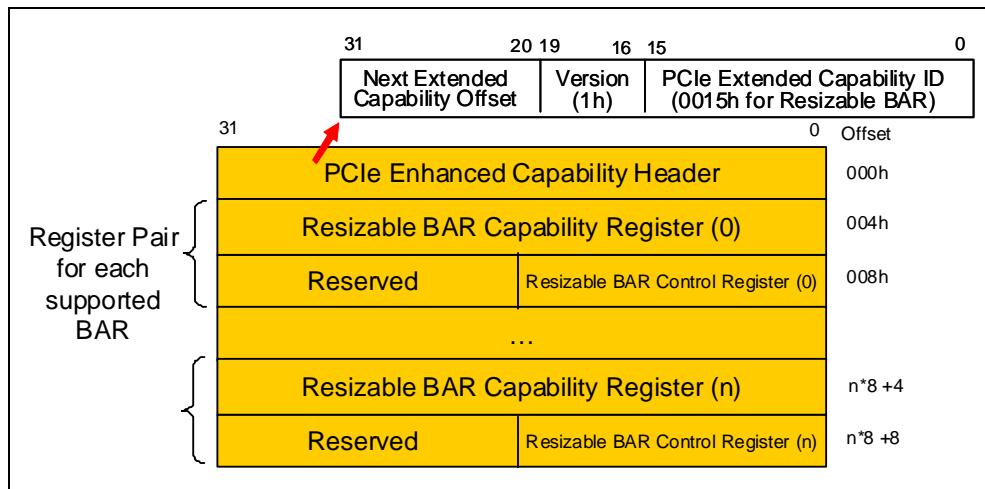
A few rules apply to the use of these registers:

1. To avoid confusion, a BAR size should only be changed when the Memory Enable bit has been cleared in the Command register.
2. The spec strongly recommends that Functions not advertise BARs that are bigger than they can effectively use.
3. To ensure optimal performance, software should allocate the biggest BAR size that will work for the system.

# PCI Express Technology

---

Figure 20-20: Resizable BAR Registers



## Capability Register

This register simply reports which BAR sizes will work for this Function. Bits 4 to 23 are used for this and the values are as shown here:

- Bit 4 - 1MB BAR size will work for this Function
- Bit 5 - 2MB
- Bit 6 - 4MB
- ...
- Bit 23 - 512GB will work for this Function

Figure 20-21: Resizable BAR Capability Register



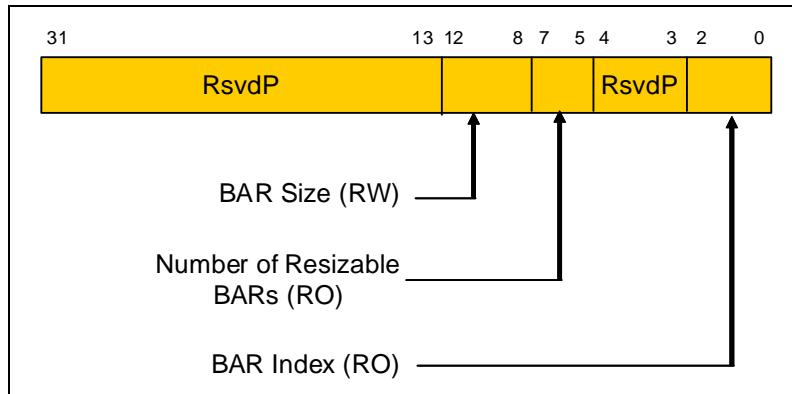
## Control Register

The BAR Index field in this register reports to which BAR this size refers (0 to 5 are possible). The Number of Resizable BARs field is only defined for Control

## Chapter 20: Updates for Spec Revision 2.1

Register zero and is reserved for all the others. It tells how many of the six possible BARs actually have an adjustable size. Finally, the BAR Size field is programmed by software to specify the desired size the BAR indicated by the BAR Index field (0 = 1MB, 1=2MB, 2=4MB, ..., 19=512GB).

Figure 20-22: Resizable BAR Control Register

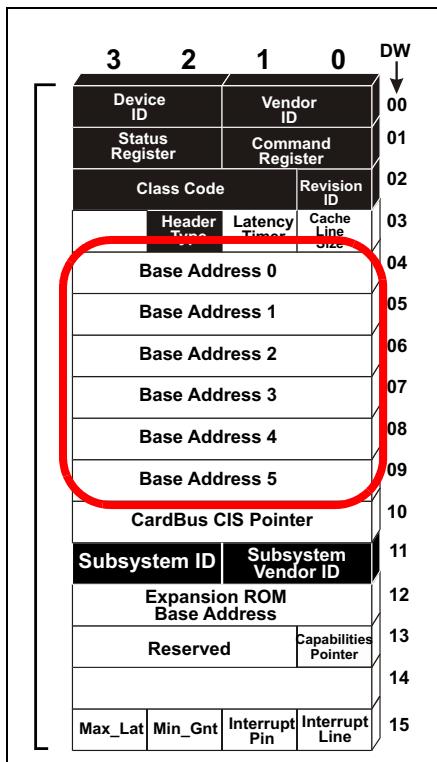


Once the Resizable values have been programmed, then enumeration software will be able to work as it normally does: writing all F's to each BAR and reading it back will report the size that was selected. Note that if the size value is changed, the contents of the BAR will be lost and will need to be reprogrammed if it was previously set up. Figure 20-23 on page 914 highlights the BAR registers in the configuration header space for a Type 0 header.

# PCI Express Technology

---

Figure 20-23: BARs in a Type0 Configuration Header



---

## Simplified Ordering Table

This change simplifies the Transaction Ordering Table by reducing the number of entries in the table. Essentially, it no longer distinguishes between completions for reads or completions for non-posted writes. The motivation for this was to reduce the number of cases that needed to be tested. For more on this, see the section called “The Simplified Ordering Rules Table” on page 288.

# Appendices



---

# *Appendix A:*

## *Debugging PCIe Traffic with LeCroy Tools*

Christopher Webb, LeCroy Corporation

---

### **Overview**

The transition of IO bus architecture from PCI to PCI Express had a large impact on developers with respect to types of tools required for validation and debug.

With parallel buses such as PCI, a waveform view of the signals shows enough information for the developer to interpret the state of the bus. A user could visually examine a waveform and mentally decode the type of transactions, how much data is transferred, and even the content of that transfer.

Since PCI Express packet traffic is both encoded and scrambled, examining a waveform view of the traffic provides very little information about the state of the link. The speed of the link can be inferred from the width of the bit times, and the width of the link can be inferred by the number of active lanes. However, the user cannot visually interpret the symbol alignment, let alone the packets themselves.

A new class of tools evolved to help developers visualize the state of their now serial links. These tools perform the de-serialization, decoding, and de-scrambling for the users. At first glance this would seem to be enough for the developer. But for PCI Express specifically, other complications such as flow control credits, lane-to-lane skew, polarity inversion, and lane reversal must also be comprehended by these tools as part of understanding PCIe protocol.

Both pre- and post-silicon debug share a common need for tools. In this appendix chapter, we describe some of the product offerings available for debugging PCIe interconnects, both from a pre and post silicon perspective.

---

## Pre-silicon Debugging

---

### RTL Simulation Perspective

In RTL simulation, looking at a waveform view of an FPGA or an ASIC signal is the most common way to debug. By showing internal state machine states, monitoring IO as it moves through the device, or seeing the state of control signals; a waveform view is quite powerful. But, as we discussed above, a PCI express link is not understandable when shown as a waveform. Additional processing or decoding must be done to make sense of this new link. To augment the simulation tools, a PCI Express Bus Monitor is typically added to address this need.

---

### PCI Express RTL Bus Monitor

A PCI Express Bus monitor is a piece of code which users insert in their RTL simulation to help monitor the state of their PCIe link. At minimum, this monitor will output text based log files with information about link state changes and types of packet activity. More complex monitors will perform real time compliance checking. A number of vendors provide purchasable IP which perform this exact function. The emphasis however is typically on compliance. Less functionality is provided with respect to visualization of things such as flow control credits, link utilization, or link training debug.

---

### RTL vector export to PETracer Application

LeCroy has partnered with a number of the leading PCIe verification IP vendors to create tools to further enhance the visualization and analysis of pre-silicon PCIe traffic. This involves using the vendors Bus Monitor to export raw symbol traffic into the same PETracer application used by the dedicated PCIe Analyzer hardware. SimPASS PE is LeCroy's solution to supporting this export.

More information about LeCroy's PETracer application and its features are described in the section "As a last resort, a flying lead probe shown in Figure 5 on page 924 may be used to attach the protocol analyzer to the system under test. This involves soldering a resistive tap circuit and connector pins to the PCIe traces. This circuitry is typically soldered to the AC coupling caps of the PCIe link as they are often the only place to access the traces. Once the probe cir-

cuity is soldered to the PCB, the analyzer probe can be connected and removed as needed. This approach can be used on virtually any PCIe link, however the robustness of the connection is limited by the skill of the technician adding the probe.” on page 924.

---

## Post-Silicon Debug

---

### Oscilloscope

Use of an oscilloscope for debugging a PCIe link is typically focused on the electrical validation of the link. The most common usage is examining an eye pattern with a mask overlay for determining electrical compliance. A lesser known compliance check is to examine the entry and exit of electrical idle state to see if the link goes to the common mode voltage within the required time periods after an electrical idle ordered set is transmitted. These are 2 examples of PCIe compliance checking which are best performed using an oscilloscope such as shown in Figure 1 on page 920.

With the addition of dynamic link training for 8.0 GT/s operation, devices must now train the transmitter emphasis during the Recovery.EQ LTSSM sub-state. The goal is to set the transmitter EQ to provide the best signal eye to the receiver. Monitoring this dynamic equalization process is another example where the use of an oscilloscope is quite powerful. With a real time oscilloscope, the user can capture this process and see the impact on the waveform as transmitter settings are changed. This allows the user to verify that the transmitter is indeed acting on the coefficient change requests, but it also allows the user to determine if the receiver has properly chosen the correct setting.

For logical debug of the link, the oscilloscope is most useful when the link is x1 or x2 as you are limited by the number channels the scope can acquire. The first method of examining PCIe traffic is a waveform view. As with the RTL waveform viewer, there is little to understand about the state of the link without SW help to perform 8b/10b decoding and de-scrambling. Fortunately, more advanced oscilloscopes have SW packages that perform these duties. For this to work properly, the scope must have deep capture buffers and must see the SKIP ordered sets so that they can decipher the byte alignment and synchronize the de-scrambler LFSR.

The LeCroy Oscilloscope can overlay PCIe symbols right onto the waveform for enhanced visibility of the traffic. An additional text based listing of the packet symbols can be displayed on the screen as an additional method of examining the waveform.

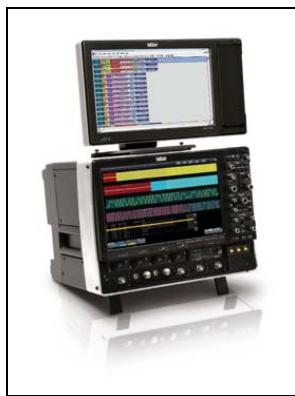
## PCI Express Technology

---

LeCroy recently announced a SW package called ProtoSync for their oscilloscope line which allows the user to export the captured waveform into the PETracer application. This is the same SW package that the protocol analyzer uses which includes a wide range of post processing capabilities described below. The PETracer software can run independently on the scope hardware, often on a second monitor. This allows time correlated comparison of the physical layer data presented by the scope waveform alongside the logic layer presentation of data presented by the PETracer software.

Capture of the 8.0 GT/s dynamic link equalization on the oscilloscope and exporting this traffic to the PETracer application is a prime example where this solution is most powerful. The user can navigate PETracer to the link training packet where the TX coefficient change request has been sent, then identify where this coefficient change was applied in the scope SW. The user can then measure the time it takes for the coefficient change to be applied and compare this to the timing required in the PCIe spec.

*Figure A-1: LeCroy Oscilloscope with ProtoSync Software Option*



---

## Protocol Analyzer

A growing trend in debugging PCIe links is to use a dedicated protocol analysis tool. What separates a protocol analyzer from a logic analyzer is that it is built to support a specific protocol such as PCIe. From a hardware perspective, a PCIe protocol analyzer is optimized for acquiring and storing PCIe traffic. This starts from the dedicated PCIe interposer probes, continues to the cabling choice, and carries through into the internal hardware components. For recovering PCIe traffic, specialized clock and data recovery circuits are used which can handle the electrical idle transitions, spread spectrum modulation, as well as

handle the run lengths found in 128b/130b encoding. Sophisticated equalization circuits are used to recover the signal eye prior to deserialization. Without comprehending the complexities of PCIe recovery, the Analyzer hardware would not be optimized for recovering complex traffic such as speed switching, dynamic link widths, and low power states such as L0s.

In addition to choosing appropriate hardware components for recovering PCIe traffic, a protocol analyzer includes logic circuitry which is PCIe specific. This logic must infer the state of the PCIe link and follow it during various LTSSM state changes. Once the link state is being properly followed, dedicated packet inspection circuits perform data matching against incoming packets to look for events programmed by the user. These matchers are used for filtering of traffic as well as performing the trigger functionality needed for stopping the traffic capture. A mixture of these traffic filters as well as deep trace buffers (often 4GB to 8GB in size) allow the user to capture significantly longer traffic scenarios than would be possible without a protocol analyzer.

Finally, the most important piece of a protocol analyzer is the software GUI. By optimizing the traffic views, post processing reports, and hardware controls with a dedicated PCI Express software tool; a very comprehensive set of PCI express specific analysis can be performed.

---

### Logic Analyzer

Some logic analyzers offer PCIe specific software packages. This software will read the PCI express capture from the logic analyzer hardware and perform some amount of post processing of this data. This analysis includes the basics such as decoding, de-scrambling, and decoding of the traffic. These SW tools do not perform many of the rich post processing features offered by dedicated protocol analyzer software, however.

---

### Using a Protocol Analyzer Probing Option

To record your PCIe traffic you must first find the best method for probing it. PCIe started as an add-in card form factor for desktop PC's and servers, but has since proliferated into a dizzying array of standard and non-standard form factors. For the standard form factors, the best probe option is a dedicated interposer.

An Interposer is a dedicated piece of hardware which includes probe circuitry required for passing a copy of the PCIe traffic to the Analyzer hardware for capture and analysis. These interposers are designed specifically for the mechanical

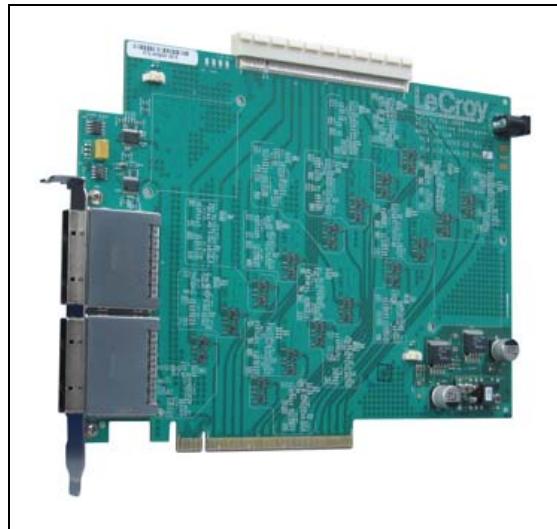
## **PCI Express Technology**

---

and electrical environments for which they are placed. The most common interposer is a “Slot Interposer” such as shown in Figure 2 on page 922. This interposer is used for probing standard CEM compliant PCIe add-in cards.

Care should be taken when selecting an interposer as the probe circuitry varies by vendor and by requirements imposed by the max PCIe link speed. For example, a Gen3 slot interposer should contain probe circuitry which allows the dynamic link training process to pass properly through the probe. The LeCroy Gen3 slot interposer uses linear circuits to maintain the shape of the waveform as it passes through the probe. This allows pre-emphasis of the transmitter to be dynamically changed during link training while allowing the receiver to quantify the impact of a new setting (either positive or negative impact).

*Figure A-2: LeCroy PCI Express Slot Interposer x16*



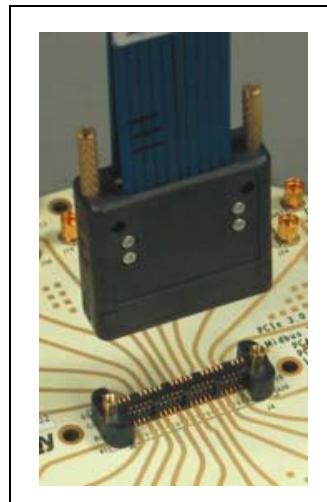
LeCroy also offers a family of other dedicated interposers for form factors such as ExpressCard, XMC, Mini Card, Express Module, AMC, etc. Some of these interposers are shown in Figure 3 on page 923. For a complete list of these interposers please refer to the LeCroy website: [www.lecroy.com](http://www.lecroy.com) as this list is constantly growing.

*Figure A-3: LeCroy XMC, AMC, and Mini Card Interposers*



For debugging PCIe links which cannot benefit from a dedicated interposer, a mid-bus probe shown in Figure 4 on page 923 is the next best option. A mid-bus probe involves placement of an industry standard probe geometry on the PCB. Each PCIe lane is routed to a pair of pads on the footprint which can be probed using a mid-bus probe head. These probes use spring pins or C clips for providing solder free mechanical attachment between the system under test and the protocol analyzer.

*Figure A-4: LeCroy PCI Express Gen3 Mid-Bus Probe*



# PCI Express Technology

---

As a last resort, a flying lead probe shown in Figure 5 on page 924 may be used to attach the protocol analyzer to the system under test. This involves soldering a resistive tap circuit and connector pins to the PCIe traces. This circuitry is typically soldered to the AC coupling caps of the PCIe link as they are often the only place to access the traces. Once the probe circuitry is soldered to the PCB, the analyzer probe can be connected and removed as needed. This approach can be used on virtually any PCIe link, however the robustness of the connection is limited by the skill of the technician adding the probe.

*Figure A-5: LeCroy PCI Express Gen2 Flying Lead Probe*



---

## Viewing Traffic Using the PETracer Application

---

### CATC Trace Viewer

The primary way to view PCI Express traffic with the LeCroy PETracer application is the CATC Trace view. This view takes each recorded packet and breaks it down into different packet fields to highlight the important values contained in this packet. A mixture of colors and text are used to visually categorize and explain the purpose of each packet. Errors are highlighted in red such as shown in Figure 6 on page 925. Warnings are highlighted in yellow making it easy to identify areas of traffic or fields in a packet which are non-compliant.

Figure A-6: TLP Packet with ECRC Error

Packet	25	Packet Error	TLP	Chg	OpType	Length	RequesterID	Tag	DeviceID	ResponseID	1st BE	ECRC	LCRC
1	x6	ECRC Err	0	00:00:100	1	000:00:0	0	000:00:0	0x000	1111	0x11223444	0xB13BC064	

In addition to decoding and visually breaking down each packet, a hierarchical display allows logical grouping of related packets. For example, in “Link Level” mode, TLP packets are grouped with their respective ACK packet. Each TLP is identified as either implicitly or explicitly ACK’d or NAK’d. An example of a ACK DLLP is shown in Figure 7 on page 925 along with the ACK’d TLP.

Figure A-7: “Link Level” Groups TLP Packets with their Link Layer Response

Link Trx	183801	00	80	TLP	Mem	MWC32	Length	RequesterID	Tag	Address	1st BE	Last BE	Data	VOID	Explict ACK	Metric	# Packets	Time Stamp	
Packet	009332	x6	00	TLP	Mem	MWC32	Length	RequesterID	Tag	Address	1st BE	Last BE	Data	LCRC	Time Delta	Time Stamp			
DLLP	009337	x6	00	ACK	Address	Req. Num	CRC 16	Time Delta											
Packet	009337	x6	00	DLLP	ACK	Address	Req. Num	CRC 16	Time Delta	0000:00:0000:0000:0000:0000:0000:0000	0000:00:0000:0000:0000:0000:0000:0000	0000:00:0000:0000:0000:0000:0000:0000	0000:00:0000:0000:0000:0000:0000:0000	0000:00:0000:0000:0000:0000:0000:0000	0000:00:0000:0000:0000:0000:0000:0000	0000:00:0000:0000:0000:0000:0000:0000	0000:00:0000:0000:0000:0000:0000:0000	0000:00:0000:0000:0000:0000:0000:0000	0000:00:0000:0000:0000:0000:0000:0000

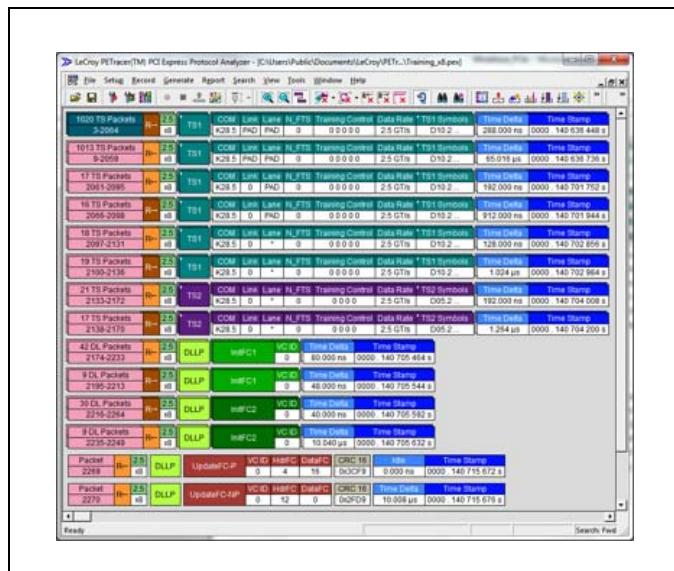
In “Split-Level” mode shown in Figure 8 on page 926, the CATC Trace view combines split transactions. For example, a single TLP read can be grouped with 1 or more completion TLPs to logically show large data transfers as a single line in the trace. The amount of data, starting address, as well as performance metrics are provided for each split level transaction. This allows the user to bypass the details of how large memory transactions are broken into multiple TLP packets and rather focus on the contents of the data. If the user wishes to see the details of the split transaction, the hierarchical display can show the link level and/or packet level breakdown of all the packets which make up this split transaction. This “drill-down” approach to traffic analysis allows the user to start from a high level view of what’s happening on the bus and drill down only in the areas of traffic which are interesting to the user.

# PCI Express Technology

*Figure A-8: "Split Level" Groups Completions with Associated Non-Posted Request*

The CATC trace view also supports “Compact-View” shown in Figure 9 on page 927. In this view, packets which are sent repeatedly are collapsed into a single cell. This is very useful for collapsing Training Sequences or Flow Control Initialization packets. The software algorithms which perform this collapse are smart enough to collapse any SKIP ordered sets as well. This creates a very compact view of the link training process allowing the user to examine changes in the link training packets without scrolling through hundreds of packets.

*Figure A-9: “Compact View” Collapses Related Packets for Easy Viewing of Link Training*



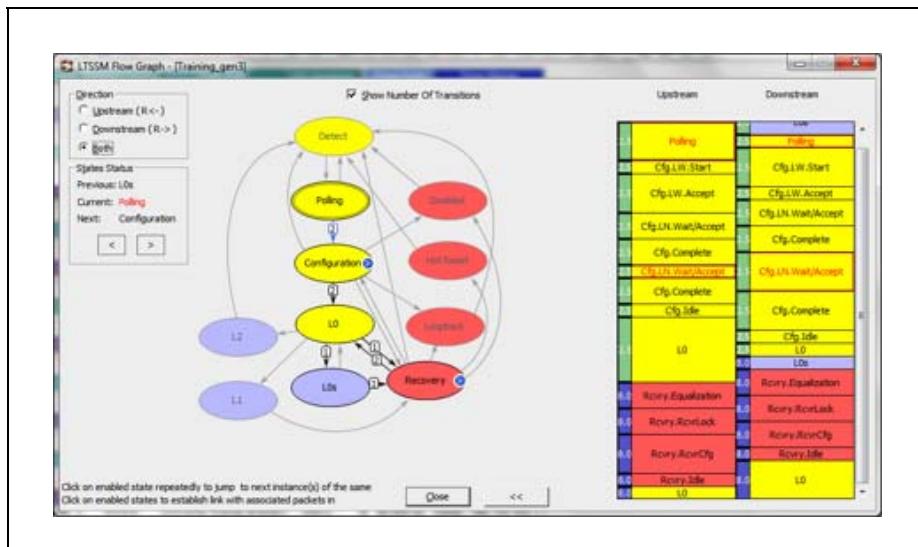
## LTSSM Graphs

To further enhance the “drill-down” traffic viewing approach, the PETracer application includes an LTSSM graph view as shown in Figure 10 on page 928. When this graph is invoked, the SW parses through the trace to find the link training sections and infers the state of the Link Training and Status State Machine (LTSSM). The result is a graph which breaks down the LTSSM state transitions in a very high level view. This graph allows the user to immediately see if the link went into a recovery state. If so, the user can easily identify which side of the link initiated the recovery, how many times it entered recovery, and even if the link speed or link width decreased because of the recovery.

The LTSSM graph is also an active link back into the trace file. For example, if the user clicks on the entry to recovery, the trace file will be navigated to that location in the trace file. This would allow the user to perhaps see if the recovery was caused by repeated NAKs or for some other reason such as loss of block alignment.

In short, when users are debugging issues related to link training, speed change, or low power state transitions, the LTSSM is affected. By examining the LTSSM graph, the user can easily identify whether these link state changes occurred, where they occurred, and navigate directly to them for faster analysis.

*Figure A-10: LTSSM Graph Shows Link State Transitions Across the Trace*



## Flow Control Credit Tracking

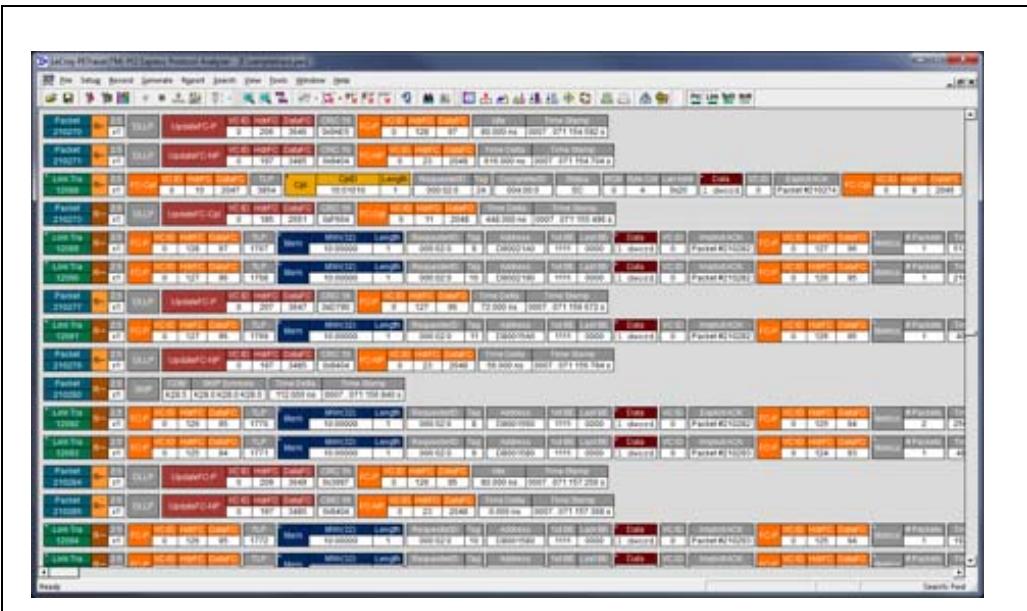
Flow control credit tracking is particularly problematic in PCI express. The flow control update packets do not show the number of credits each endpoint has, rather it shows how many credits in total have been used. This means that each endpoint must keep a running counter of credits for each type. There are a number of scenarios where credits can be lost, and if this occurs, the link will eventually be unable to transmit data due to lack of credits. Such problems are very difficult to identify and debug.

The LeCroy PETracer application has a credit tracking SW tool shown in Figure 11 on page 929 to aid in this debug. If the trace contains FC-Init packets, it will walk through the trace and show the amount of remaining credits per virtual channel buffer type after each TLP and FC-Update.

FC-Init packets are sent once after link training. Because of this, the PETracer application has the ability for the user to set initial credit values at some point in

the trace and the SW will calculate the relative credit values for the remaining packets. Even if the initial credit values are set improperly by the user, having the ability to see the relative credits is often enough to catch a flow control issue.

Figure A-11: Flow Control Credit Tracking



## Bit Tracer

Some debug situations are not solved by a drill down approach to examining the traffic. For example if the link settings are incorrect, the recording is often unreadable. What if a device is not properly scrambling the traffic, or the 10 bit symbols are sent in reverse order? For this scenario, a tool which focuses on analysis between the waveform view of the scope and the CATC Trace view is needed. This is where the BitTracer view shown in Figure 12 on page 930 is most powerful.

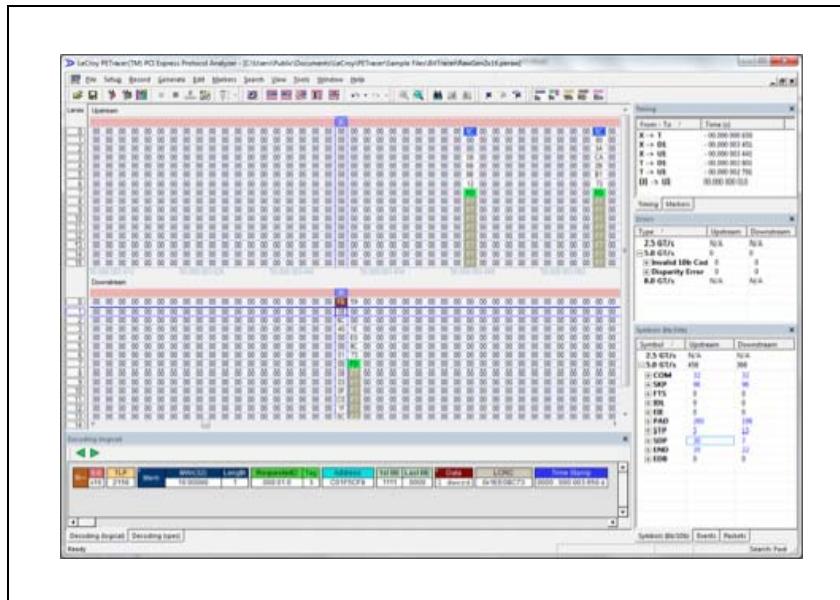
The BitTracer view allows the user to see raw traffic exactly as it was seen on the link. The software allows the user to see the traffic as 10 bit symbols, scrambled bytes, or unscrambled bytes. Invalid symbols and incorrect running disparity are highlighted in red.

# PCI Express Technology

---

To further determine what may be wrong with the traffic, the BitTracer tool adds a powerful list of post processing features which can modify the traffic. For example, post capture; the user can invert the polarity of a given lane. Once applied, the user can see if the 10 bit symbols are now represented properly in the trace. If this cleans up the trace, it's an indication that the recording settings for the Analyzer hardware need to be changed.

Figure A-12: BitTracer View of Gen2 Traffic



In addition, the lane ordering can be modified. This is useful for determining if lane reversal is causing a bad capture. If the traffic has excessive lane to lane skew, the BitTracer software allows the user to re-align the traffic. For Gen3 traffic, this skew can be applied 1 bit at a time. This essentially allows the user to fix the 130 bit block alignment post capture.

After applying changes to the data, all or just a portion of the data can be exported into the standard CATC Trace view for higher level analysis. This workflow is very powerful for debugging low level issues during early bring-up. Let's say for example, the user's device trains the link properly, and then suddenly applies polarity inversion to 1 lane. This is a clear violation of the spec and will cause the link to retrain. If this traffic is captured with the BitTracer tool, the user could easily identify this as the problem. Additionally, the portion of the traffic before and after the inversion could be exported into separate trace files and examined in the CATC Trace view.

---

### Analysis overview

As you can see, different traffic views can be beneficial for debugging certain failure conditions. LeCroy supports import of PCIe traffic from many sources into its highly sophisticated PETracer software. Whether the source is RTL simulation, an oscilloscope capture, or a dedicated protocol analyzer capture, PETracer has a rich set of traffic views and reports which allow the user to best understand the health and state of their PCIe link.

---

## Traffic generation

---

### Pre-Silicon

For stimulating a PCI Express endpoint in simulation, dedicated verification IP can be purchased from a number of vendors. This IP will test for basic functionality as well as perform a number of PCIe compliance checks. It is certainly in the interest of the ASIC developer to find and fix these issues before tapeout, and this is where the value of these tools comes from. If the PCIe design is implemented in an FPGA where mask costs are not an issue, it may be more cost effective to perform these compliance checks in hardware with a dedicated traffic generation tool such as the LeCroy PETrainer or LeCroy PTC card.

---

### Post-Silicon

#### Exerciser Card

To thoroughly test the PCIe compliance and overall robustness of a PCIe design post-silicon, a dedicated Exerciser card such as the LeCroy PETrainer shown in Figure 13 on page 932 is used. This card allows the user to generate a wide range of compliant and non-compliant traffic. For example, if you place your PCIe card in a standard motherboard, you may be limited in the size of the TLP packets it will see. A dedicated Exerciser card can generate TLP packets across the entire legal range of packet sizes.

Secondly, if you would like to test that a card issues a NAK in response to a TLP with a bad LCRC, it would not possible with the card connected to compliant devices. They do not transmit bad packets. An Exerciser card can create a TLP with a bad LCRC, improper header values, or end the TLP with an EDB symbol.

# PCI Express Technology

---

If you would like to test that your card properly replay's a packet when it receives a NAK, this can be done with an Exerciser. Perhaps you would like to issue 4 NAKs in a row to a certain TLP so that link recovery is initiated. This behavior is all quite easy to program into the exerciser card.

The number of test cases and failure scenarios is limited only by the number of scripts you write. Once written, these scripts can be re-used for testing new versions of your design. The Analyzer SW can record these sessions and use scripting to determine if the response was correct. A number of LeCroy customers have created large libraries of regression tests using these tools.

*Figure A-13: LeCroy Gen3 PETrainer Exerciser Card*

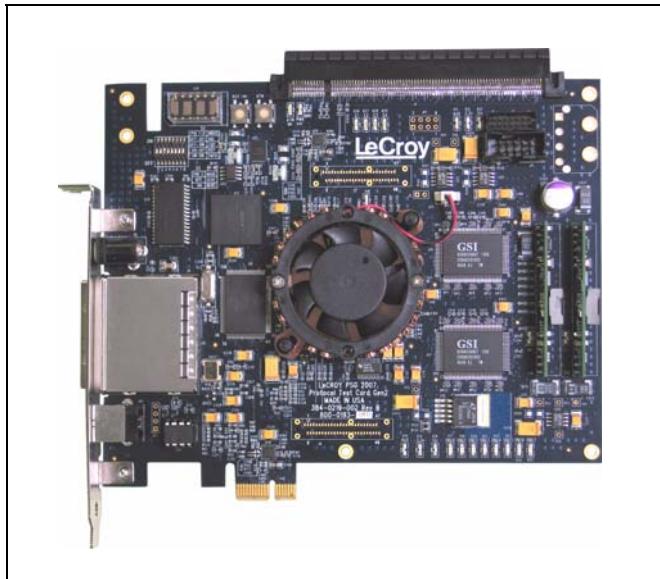


## PTC card

The PCI SIG has published a specific list of compliance tests which all "Compliant" devices must pass. The LeCroy Protocol Test Card (PTC) is the hardware used to perform these tests at the PCI SIG Compliance workshops. Users can purchase a PTC card from LeCroy shown in Figure 14 on page 933 to pre-test their devices to ensure they will pass PCI SIG compliance testing.

The LeCroy PTC is used to test root complex or endpoint devices at x1 link widths. Link speeds can be either Gen1 or Gen2.

Figure A-14: LeCroy Gen2 Protocol Test Card (PTC)



---

## Conclusion

Today, the PCIe developer has access to a wide range of tools to help debug their PCIe design. Thanks to the wide adoption of the PCIe standard, many of these tools are designed specifically for PCIe debug and include features which address the challenges many PCIe devices face.

For more information about the LeCroy PCIe tool offerings, please visit the LeCroy website [www.lecroy.com](http://www.lecroy.com)

# PCI Express Technology

---

# *Appendix B:*

## *Markets & Applications for PCI Express*

Akber Kazmi (Senior Director Marketing, PLX Technology, Inc.)

---

### **Introduction**

Since its definition in the early 1990s, PCI has emerged as the most successful interconnect technology ever used in computers. Originally intended for personal computer systems, the PCI architecture has expanded into virtually every computing platform category, including servers, storage, communications, and a wide range of embedded control applications. Most important, each advancement in PCI bus speed and width provided backward compatibility.

As successful as the PCI architecture was, there was a limit to what could be accomplished with a multi-drop, parallel, shared-bus interconnect technology. A number of issues -- clock skew, high pin count, trace routing restrictions in printed circuit boards (PCB), bandwidth and latency requirements, physical scalability, and the need to support Quality of Service (QoS) within a system for a wide variety of applications -- lead to the definition of the PCI Express<sup>a</sup> (PCIe) architecture.

PCIe was the natural successor to PCI, and was developed to provide the advantages of a state-of-the-art, high-speed serial interconnect technology with a packet-based layered architecture, but maintain backward-compatibility with the large PCI software infrastructure. The key goal was to provide an optimized, universal interconnect solution for a wide variety of future platforms, including desktop, server, workstation, storage, communications, and embedded systems.

# PCI Express 3.0 Technology

After its introduction in 2001, PCIe has gone through three generations of enhancements. In the first generation (Gen1), signaling rate was set at 2.5 GT/s and later enhanced to 5 GT/s (Gen2) and eventually 8 GT/s (Gen3). The PCIe specification allows combining of 2, 4, 8, 12, 16 or 32 lanes into a single port. However, products available today do not support 12- and 32-lane-wide ports. It is important to note that all PCIe Gen2 and Gen3 devices are required to be backward-compatible in speed with that of the previous generation.

The industry has launched and has fully embraced PCIe Gen3 products, while at the same time the PCI Special Interest Group (PCI-SIG) is analyzing signaling rate (speed) for Gen4. The goal for PCIe Gen4 is to double the speed of Gen3, to 16 GT/s.

PCIe switches are available in an array of sizes, ranging from 3 to 96 lanes, and 3 to 24 ports where each port could be one, two, four, eight or 16 lanes wide. A Gen3 single lane would provide 1GB/s of bandwidth, while a 16-lane port offers 16GB bandwidth in each direction. Additionally, PCIe switch vendors, such as PLX Technology, have added features and enhancement to their products that are not part of PCIe specifications but enable them to differentiate their products and add value for the system designers. These features deliver ease of use, higher performance, fail-over, error detection, error isolation, and field-upgradability.

On-chip features include non-transparent (NT) bridging, peer-to-peer communication, Hot-Plug, direct memory access (DMA), and error checking/recovery. Additionally debug features such as packet generation, receiver-eye measurement, traffic monitoring, and error injection in live traffic offer significant value to the designers, enabling early system bring-up. Many of these features can also be used for run-time performance improvements and monitoring.

Features included in next generation of PCIe switches are:

- **NT bridging:** Allows two hosts/CPUs to be connected to the same PCIe switch while electrically and logically isolated. The NT bridging functions is broadly used in systems requiring isolation of two active CPUs or two CPUs where one is active and other is passive. The NT functionality allows the exchange of heartbeat between the two host CPUs to enable fail-over if one of them fails.

## **Chapter : Appendix B: Markets & Applications for PCI**

---

- **DMA:** An on-chip DMA controller in a PCIe switch offers significant value to the designers as it enables them to spare CPU cycles to move data between peers and the CPU to/from I/Os. The CPU's reduced effort in moving data boosts overall performance of the system as the spared CPU cycles can be used to run applications rather than managing data I/O.
- **Error Isolation:** Users can program triggers for certain error events and response by the switch. The response of switch can also be programmed to ignore, trigger a host interrupt, bring the port with errors down, or bring the entire switch down.
- **Packet Generation:** Generally, it is difficult to generate traffic that saturates a PCIe port without the use of expensive packet generator equipment. PCIe switches now have the ability to saturate any PCIe port with desired traffic, such as transaction layer packets, to check the performance and robustness of the system.

---

### **PCI Express IO Virtualization Solutions**

The PCIe technology was initially defined as a single-host interconnect technology but in last few years new standards have been developed that make PCIe suitable for multi-host systems as a switch fabric technology for data centers and enterprise IT applications. The presence of native PCIe interfaces (ports) on x86 CPUs and servers platforms has enabled designers to use PCIe as backplane and fabric technology for small to mid-size server clusters.

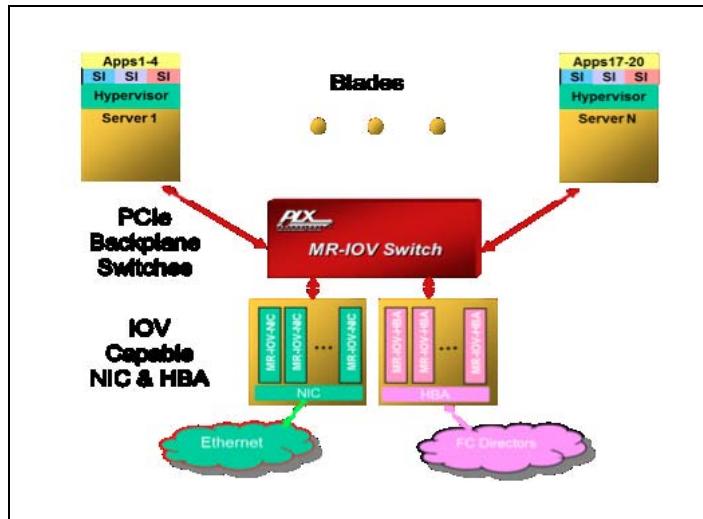
In 2007, the PCI-SIG released the Single-Root I/O Virtualization (SR-IOV) specification that enables sharing of a single physical resource such as a network interface card or host bus adapter in a PCIe system among multiple virtual machines running on one host. This is the simplest approach to sharing resources or I/O devices among different applications or virtual machines.

The PCI-SIG followed by completing, in 2008, work on its Multi-Root I/O Virtualization (MR-IOV) specification that extends the use of PCIe technology from a single-root domain to a multi-root domain. The MR-IOV specification enables the use of a single I/O device by multiple hosts and multiple system images simultaneously, as illustrated in Figure 0-1 on page 938. This illustration shows a multi-host environment where MR-IOV capable NIC and HBA are shared across multiple servers or virtual machines via an MR-IOV switch.

# PCI Express 3.0 Technology

---

Figure 0-1: MR-IOV Switch Usage



In order to implement MR-IOV specifications, three components of the system need to be developed – MR-IOV PCIe switches, endpoints, and management software. All three of these components must be available simultaneously and work seamlessly. Unfortunately, four years after the specification was developed, there is not a single silicon vendor that has MR-IOV capable PCIe switch or end-points. PCIe switch vendors are offering solutions that provide capabilities defined for MR-IOV through vendor-defined features and utilizing available SR-IOV end-points.

---

## Multi-Root (MR) PCIe Switch Solution

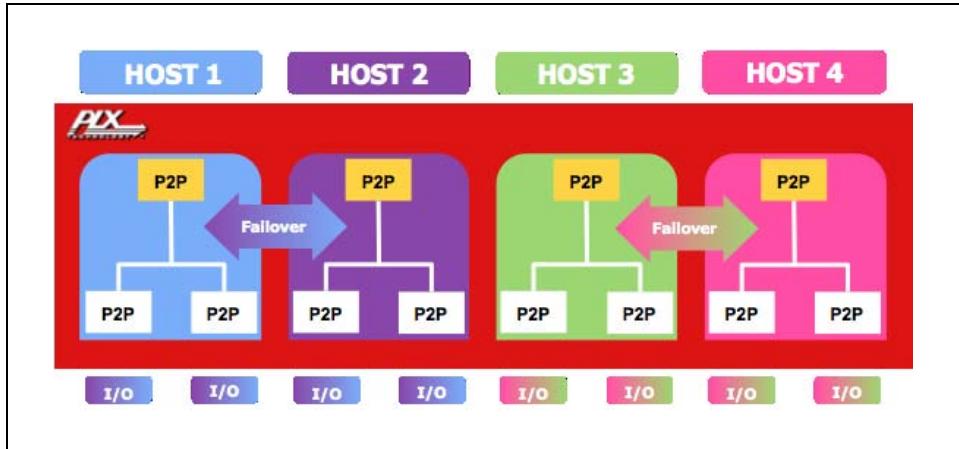
PCIe switch vendors have created switches that offer implementation of multi-host function through non-transparent bridging and multi-root (MR) capabilities. These MR switches allow multiple hosts to be connected to a single switching device, which can be portioned under user control in such a way that each host will be connected to a desired set of downstream ports of the switch.

In the MR switches, one of the hosts acts as the master and assigns I/Os to other host ports. Each host operates independently of other hosts and controls downstream devices in its domain. Figure 0-2 on page 939 illustrates the internal architecture of an MR switch, in which particular sets of downstream ports are associated to particular host ports under management control.

## **Chapter : Appendix B: Markets & Applications for PCI**

---

*Figure 0-2: MR-IOV Switch Internal Architecture*



---

### **PCIe Beyond Chip-to-Chip Interconnect**

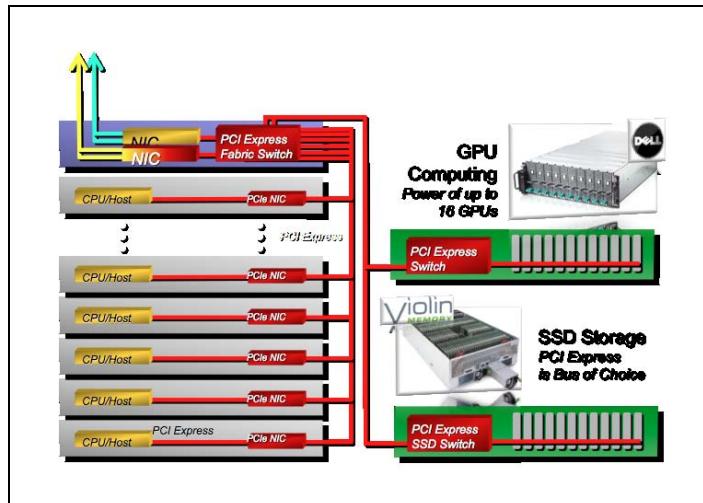
In early stages of PCIe deployments the technology was used as a chip-to-chip interconnect but now broad availability of PCIe interfaces on CPUs, chipsets and IOs and broad adoption of these components is pushing it beyond traditional applications. In a new generation of applications, PCIe is used in system backplanes, switch fabrics, cabling systems, storage/IO expansion, IO virtualization, high-performance computing (HPC), and server clusters. Figure 0-3 on page 940 illustrates use of PCIe in a data center for high performance compute application where servers in a rack are clustered through a top-of-rack (TOR) PCIe switch fabric box. The TOR PCIe switch can be connected to the network through Ethernet and to local storage and compute resources through PCIe links.

PCIe connections out-side the box depend on PCIe copper or optical cables that the leader in the industry are introducing at lower cost. The PCIe TOR fabric is suitable for server/compute clustering and may replace InfiniBand as the eco-system for PCIe as fabric grows.

# PCI Express 3.0 Technology

---

Figure 0-3: PCIe in a Data Center for HPC Applications



---

## SSD/Storage IO Expansion Boxes

Recently, the industry has converged towards PCIe as the unified interconnect technology for enterprise storage and solid state drive (SSD) applications. The NVM HCI, an industry consortium, has released a specification called NVM Express (NVMe) that uses PCIe to provide the bandwidth needed for SSD applications. Additionally, a T10 committee has embarked on defining SCSI over PCIe (SOP) protocol to take advantage of PCIe technology capabilities for high-performance storage applications. Furthermore, the SATA consortium recently announced that it would use PCIe as the interconnect for its next-generation SATA specification called SATA Express (SATAe).

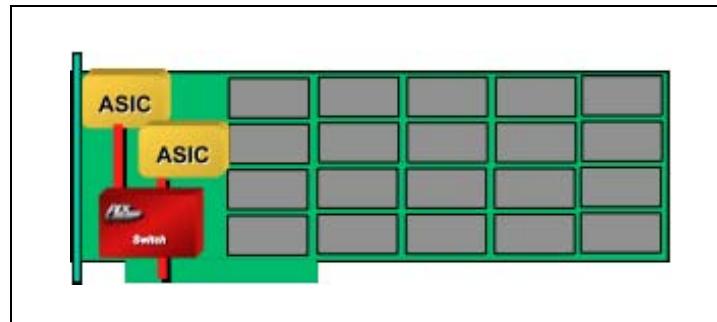
---

## PCIe in SSD Modules for Servers

Traditionally, enterprise SSD modules have shipped with SAS, SATA and Fibre Channel interfaces but due to the above-mentioned developments, a large majority of SSD controller, module and system suppliers have introduced products with PCIe interfaces. Most SSD controllers peak their performance and capacity due to a heavy load of managing flash. In high-performance applications, multiple SSD controllers (or ASICs) are used and aggregated through a PCIe switch. Figure 0-4 on page 941 shows a basic usage of a PCIe switch in an SSD add-in card that applies to any card or module form factor.

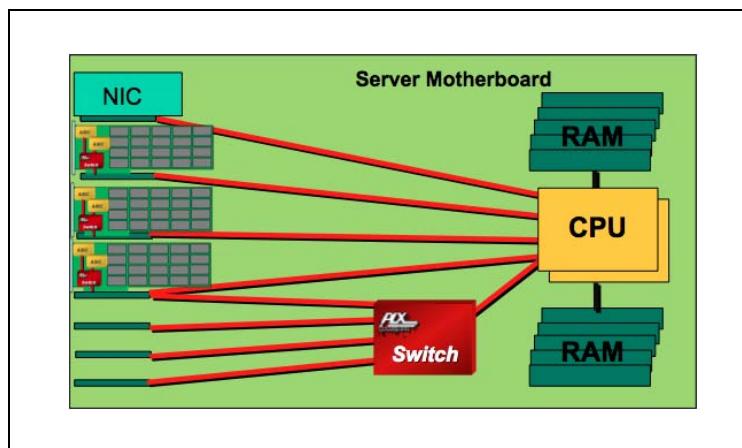
## Chapter : Appendix B: Markets & Applications for PCI

Figure 0-4: PCIe Switch Application in an SSD Add-In Card



For large data center applications, the SSD add-in cards are installed in server motherboards as shown in Figure 0-5 on page 941 and IO expansion boxes (Figure 6) aggregated through PCIe switches. In server motherboard designs, PCIe switches are utilized to create more ports/slots that accommodate additional SSD modules to support the application's needs.

Figure 0-5: Server Motherboard Use PCIe Switches

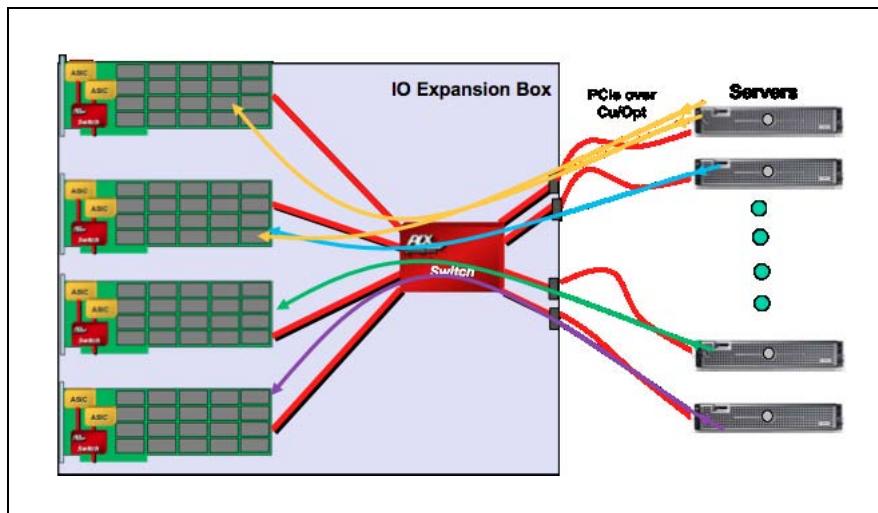


In addition to providing connectivity, PCIe switches can be used for providing redundancy and failover through NT bridging and MR functionality. The MR switches support 1+N failover capability, in which one server/host communicates with N number of servers to check the heartbeat and initiate a failover if one of them fails. One of the servers illustrated in Figure 0-6 on page 942 can be used as backup for the others in 1+N failover scheme.

# PCI Express 3.0 Technology

---

Figure 0-6: Server Failover in 1 + N Failover Scheme



---

## Conclusion

PCIe interconnect technology has become a serious contender for many high-end applications beyond chip-to-chip interconnect and is expected to be utilized in external I/O sharing, server clustering, I/O expansion and TOR switching. The current 8 GT/s and next-generation (Gen4) 16 GT/s line rates, the ability to aggregate multiple lanes in single high-bandwidth ports, fail-over capabilities, embedded DMA for data transfers, and IO sharing/virtualization provide capabilities that are at least equal to, if not superior to, interfaces such as InfiniBand and Ethernet.

---

# *Appendix C:*

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

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

---

### **Introduction**

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

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

# **PCI Express 3.0 Technology**

---

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

---

## **Usage Models**

---

### **Intelligent Adapters**

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

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

---

### **Host Failover**

Host failover capabilities are designed into systems that require high availability. High availability has become an increasingly important requirement, especially in storage and communication platforms. The only practical way to ensure that the overall system remains operational is to provide redundancy for

## **Chapter : Appendix C: Implementing Intelligent Adapt-**

---

all components. Host failover systems typically include a host based system attached to several endpoints. In addition, a backup host is attached to the system and is configured to monitor the system status. When the primary host fails, the backup host processor must not only recognize the failure, but then take steps to assume primary control, remove the failed host to prevent additional disruptions, reconstitute the system state, and continue the operation of the system without losing any data.

---

### **Multiprocessor Systems**

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

---

### **The History Multi-Processor Implementations Using PCI**

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

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

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

## **PCI Express 3.0 Technology**

---

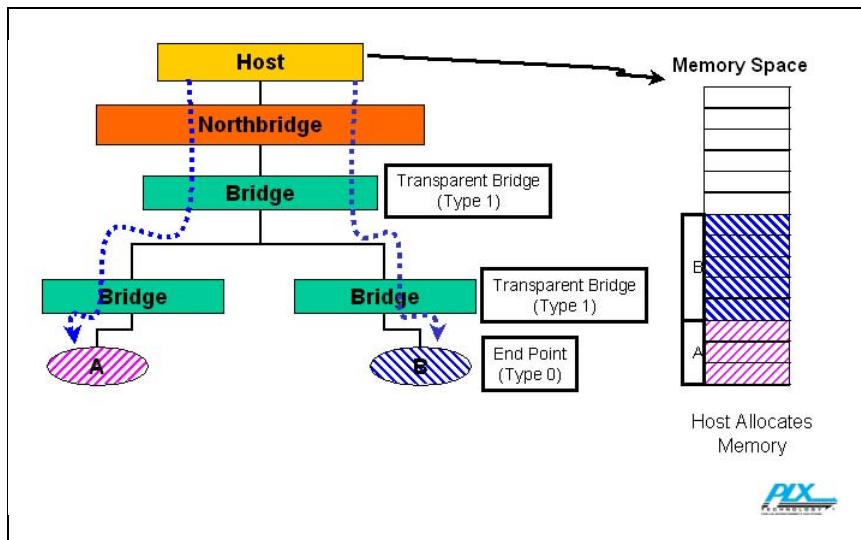
Because the host does not know the system topology when it is first powered up or reset, it must perform discovery to learn what devices are present and then map them into the memory space. To support standard discovery and configuration software, the PCI specification defines a standard format for Control and Status Registers (CSRs) of compliant devices. The standard PCI-to-PCI bridge CSR header, called a Type 1 header, includes primary, secondary and subordinate bus number registers that, when written by the host, define the CSR addresses of devices on the other side of the bridge. Bridges that employ a Type 1 CSR header are called transparent bridges.

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

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

## Chapter : Appendix C: Implementing Intelligent Adapt-

Figure 0-1: Enumeration Using Transparent Bridges



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

Up to this point, our discussions have been limited to one processor with one memory space. As technology progressed, system designers began developing end points with their own native processors built in. The problem that this caused was that both the host processor and the intelligent adapter would, upon power up or reset, attempt to enumerate the entire system, causing system conflict and ultimately a non-functional system.<sup>1</sup>

---

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

## PCI Express 3.0 Technology

---

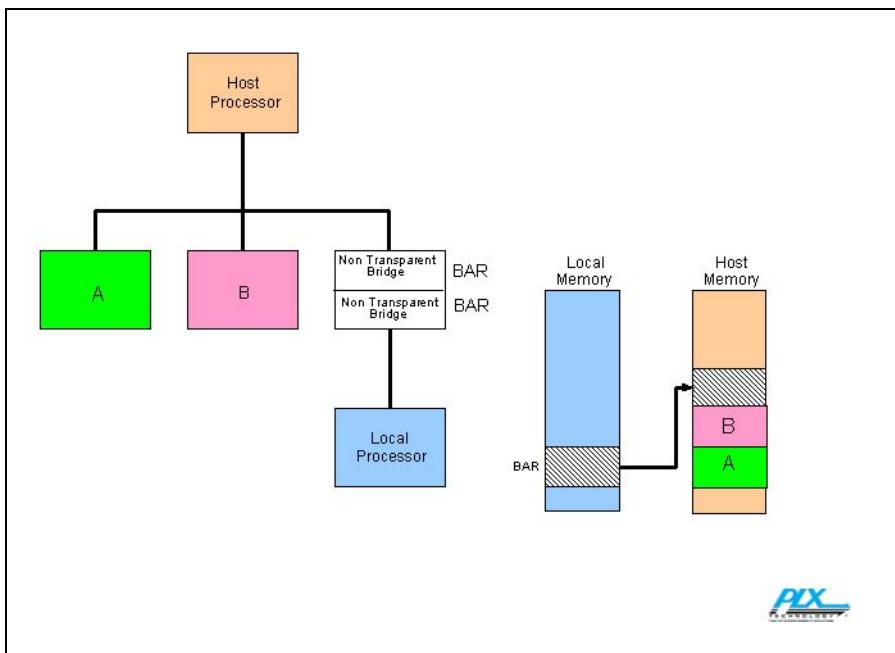
To get around this, architects designed non-transparent bridges. A non-transparent PCI-to-PCI Bridge, or PCI Express-to-PCI Express Bridge, is a bridge that exposes a Type 0 CSR header on both sides and forwards transactions from one side to the other with address translation, through apertures created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

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

Because we now have two memory spaces, the PCI Express system needs to translate addresses of transactions that cross from one memory space to the other. This is accomplished via Translation and Limit Registers associated with the BAR. See “Address Translation” on page 958 for a detailed description; Figure C-0-2 on page 949 provides a conceptual rendering of Direct Address Translation. Address translation can be done by Direct Address Translation (essentially replacement of the data under a mask), table lookup, or by adding an offset to an address. Figure C-0-3 on page 950 shows Table Lookup Translation used to create multiple windows spread across system memory space for packet originated in a local I/O processor’s domain, as well as Direct Address Translation used to create a single window in the opposite direction.

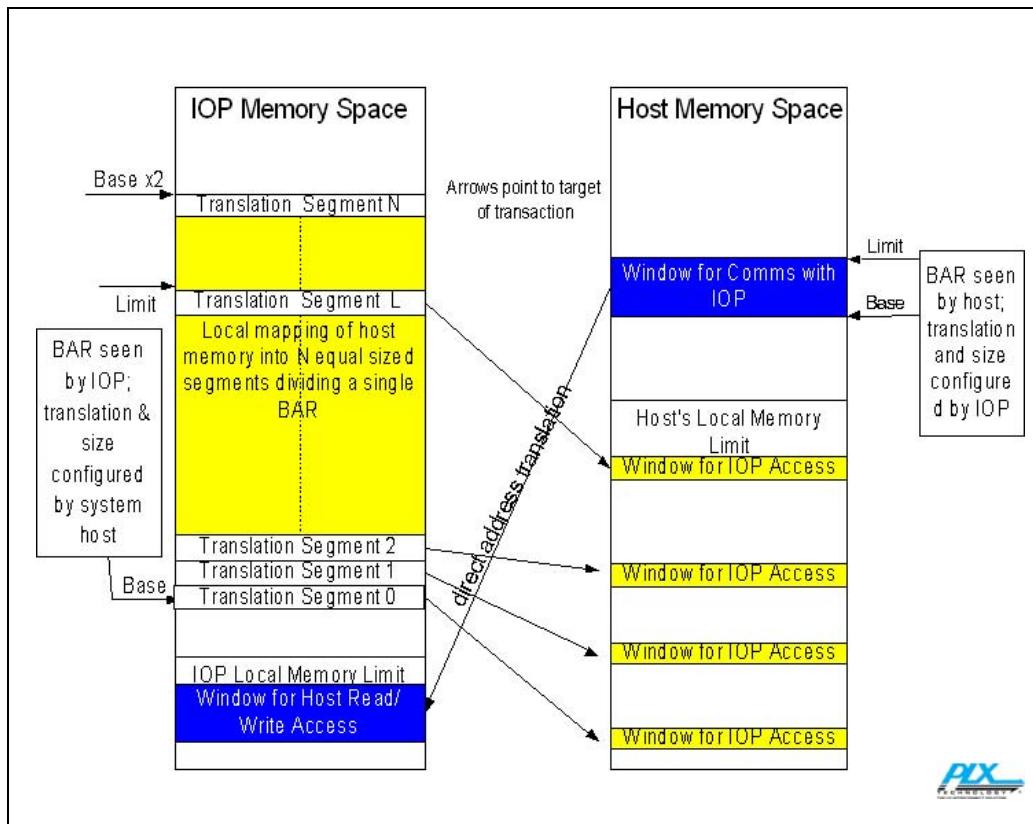
## Chapter : Appendix C: Implementing Intelligent Adapt-

Figure 0-2: Direct Address Translation



# PCI Express 3.0 Technology

Figure 0-3: Look Up Table Translation Creates Multiple Windows



## Example: Implementing Intelligent Adapters in a PCI Express Base System

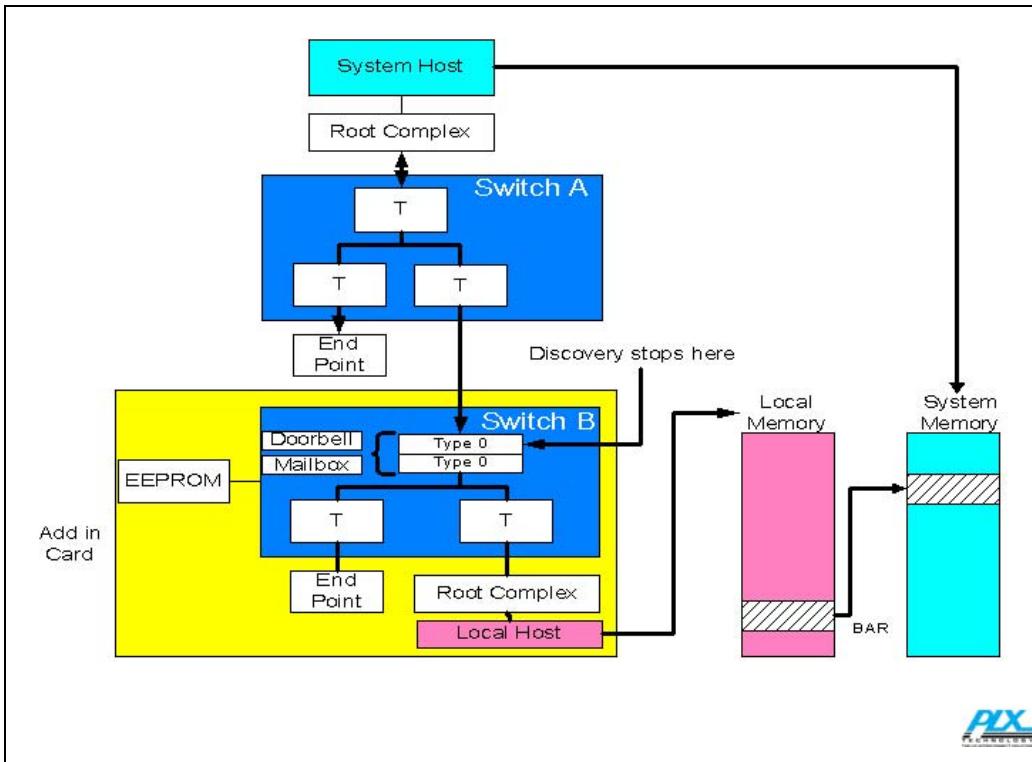
Intelligent adapters will be pervasive in PCI Express systems, and will likely be the most widely used example of systems with “multiple processors”.

Figure C-0-4 on page 951 illustrates how PCI Express systems will implement intelligent adapters. The system diagram consists of a system host, a root complex (the PCI Express version of a Northbridge), a three port switch, an example endpoint, and an intelligent add-in card. Similar to the system architecture, the add-in card contains a local host, a root complex, a three port switch, and an

## Chapter : Appendix C: Implementing Intelligent Adapt-

example endpoint. However we should note two significant differences: the intelligent add-in card contains an EEPROM, and one port of the switch contains a back to back non-transparent bridge.

Figure 0-4: Intelligent Adapters in PCI and PCI Express Systems



Upon power up, the system host will begin enumerating to determine the topology. It will pass through the Root Complex and enter the first switch (Switch A). Upon entering the topmost port, it will see a transparent bridge, so it will know to continue to enumerate. The host will then poll the leftmost port and, upon finding a Type 0 CSR header, will consider it an endpoint and explore no deeper along that branch of the PCI hierarchy. The host will then use the information in the endpoint's CSR header to configure base and limit registers in bridges and BARs in endpoints to complete the memory map for this branch of the system.

## **PCI Express 3.0 Technology**

---

The host will then explore the rightmost port of Switch A and read the CSR header registers associated with the top port of Switch B. Because this port is a non-transparent bridge, the host finds a Type 0 CSR header. The host processor therefore believes that this is an endpoint and explores no deeper along that branch of the PCI hierarchy. The host reads the BARs of the top port of Switch B to determine the memory requirements for windows into the memory space on the other side of the bridge. The memory space requirements can be preloaded from an EEPROM into the BAR Setup Registers of Switch B's non-transparent port or can be configured by the processor that is local to Switch B prior to allowing the system host to complete discovery.

Similar to the host processor power up sequence, the local host will also begin enumerating its own system. Like the system host processor, it will allocate memory for end points and continue to enumerate when it encounters a transparent bridge. When the host reaches the topmost port of Switch B, it sees a non-transparent bridge with a Type 0 CSR header. Accordingly, it reads the BARs of the CSR header to determine the memory aperture requirements, then terminates discovery along this branch of its PCI tree. Again, the memory aperture information can be supplied by an EEPROM, or by the system host.

Communication between the two processor domains is achieved via a mailbox system and doorbell interrupts. The doorbell facility allows each processor to send interrupts to the other. The mailbox facility is a set of dual ported registers that are both readable and writable by both processors. Shared memory mapped mechanisms via the BARs may also be used for inter-processor communication.

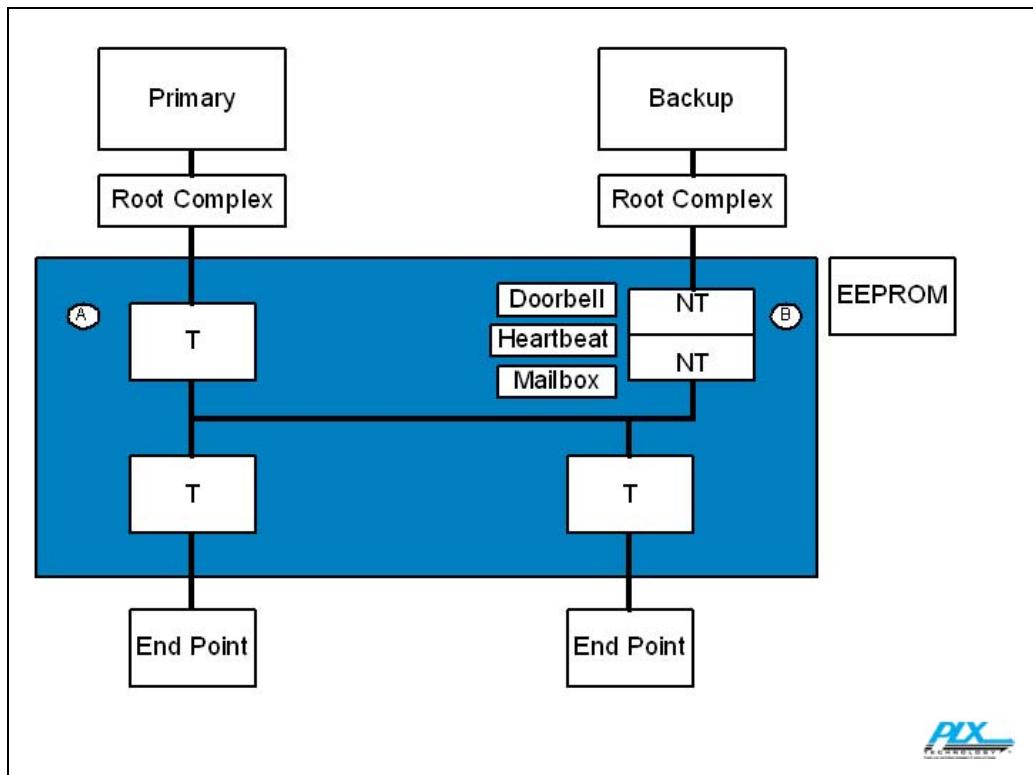
---

### **Example: Implementing Host Failover in a PCI Express System**

Figure C-0-5 on page 953 illustrates how most PCI Express systems will implement host failover. The primary host processor in this illustration is on the left side of the diagram, with the backup host on the right side of the diagram. Like most systems with which we are familiar, the host processor connects to a root complex. In turn, the root complex routes its traffic to the switch. In this example, the switch has two ports to end points in addition to the upstream port for the primary host we have just described. Furthermore, this system also has another processor, which is connected to the switch via another root complex.

## Chapter : Appendix C: Implementing Intelligent Adapt-

Figure 0-5: Host Failover in PCI and PCI Express Systems



The switch ports to both processors need to be configurable to behave either as a transparent bridge or a non-transparent bridge. An EEPROM or strap pins on the switch can be used to initially bootstrap this configuration.

Under normal operation, upon power up, the primary host begins to enumerate the system. In our example, as the primary host processor begins its discovery protocol through the fabric, it discovers the two end points, and their memory requirements, by sizing their BARs. When it gets to the upper right port, it finds a Type 0 CSR header. This signifies to the primary host processor that it should not attempt discovery on the far side of the associated switch port. As in the previous example, the BARs associated with the non-transparent switch port may have been configured by EEPROM load prior to discovery or might be configured by software running on the local processor.

## **PCI Express 3.0 Technology**

---

Again, similar to the previous example, the backup processor powers up and begins to enumerate. In this example, the backup processor chipset consists of the root complex and the backup processor only. It discovers the non-transparent switch port and terminates its discovery there. It is keyed by EEPROM loaded Device ID and Vendor ID registers to load an appropriate driver.

During the course of normal operation, the host processor performs all of its normal duties as it actively manages the system. In addition, it will send messages to the backup processor called heartbeat messages. Heartbeat messages are indications of the continued good health of the originating processor. A heartbeat message might be as simple as a doorbell interrupt assertion, but typically would include some data to reduce the possibility of a false positive. Checkpoint and journal messages are alternative approaches to providing the backup processor with a starting point, should it need to take over. In the journal methodology, the backup is provided with a list or journal of completed transactions (in the application specific sense, not in the sense of bus transactions). In the checkpoint methodology, the backup is periodically provided with a complete system state from which it can restart if necessary. The heartbeat's job is to provide the means by which the backup processor verifies that the host processor is still operational. Typically this data provides the latest activities and the state of all the peripherals.

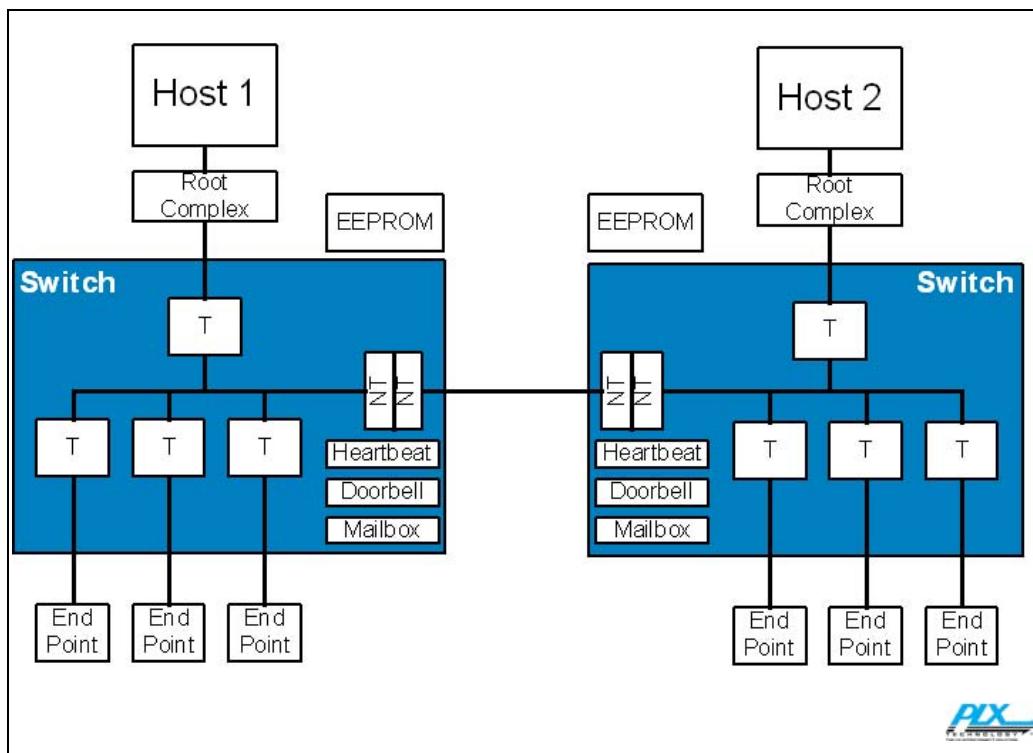
If the backup processor fails to receive timely heartbeat messages, it will begin assuming control. One of its first tasks is to demote the primary port to prevent the failed processor from interacting with the rest of the system. This is accomplished by reprogramming the CSRs of the switch using a memory mapped view of the switch's CSRs provided via a BAR in the non-transparent port. To take over, the backup processor reverses the transparent/non-transparent modes at both its port and the primary processor's port and takes down the link to the primary processor. After cleaning up any transactions left in the queues or left in an incomplete state as a result of the host failure, the backup processor reconfigures the system so that it can serve as the host. Finally, it uses the data in the checkpoint or journal messages to restart the system.

## Chapter : Appendix C: Implementing Intelligent Adapt-

### **Example: Implementing Dual Host in a PCI Express Base System**

Figure C-0-6 on page 955 illustrates how PCI Express systems might implement a dual host system<sup>1</sup>. In this example, the leftmost blocks are a typically complete system, with the rightmost blocks being a separate subsystem. As previously discussed, connecting the leftmost and rightmost diagram is a set of non-transparent bridges.

*Figure 0-6: Dual Host in a PCI and PCI Express System*



Upon power up, both processors will begin enumerating. As before, the hosts will search out the endpoints by reading the CSR and then allocate memory

1. Back to back non-transparent (NT) ports are unnecessary but occur as a result of the use of identical single board computers for both hosts. A transparent backplane fabric would typically be interposed between the two NT ports.

## **PCI Express 3.0 Technology**

---

appropriately. When the hosts encounter the non-transparent bridge port in each of their private switches, they will assume it is an endpoint and, using the data in the EEPROM, allocate resources. Both systems will use the doorbell and mailbox registers described above to communicate with each other.

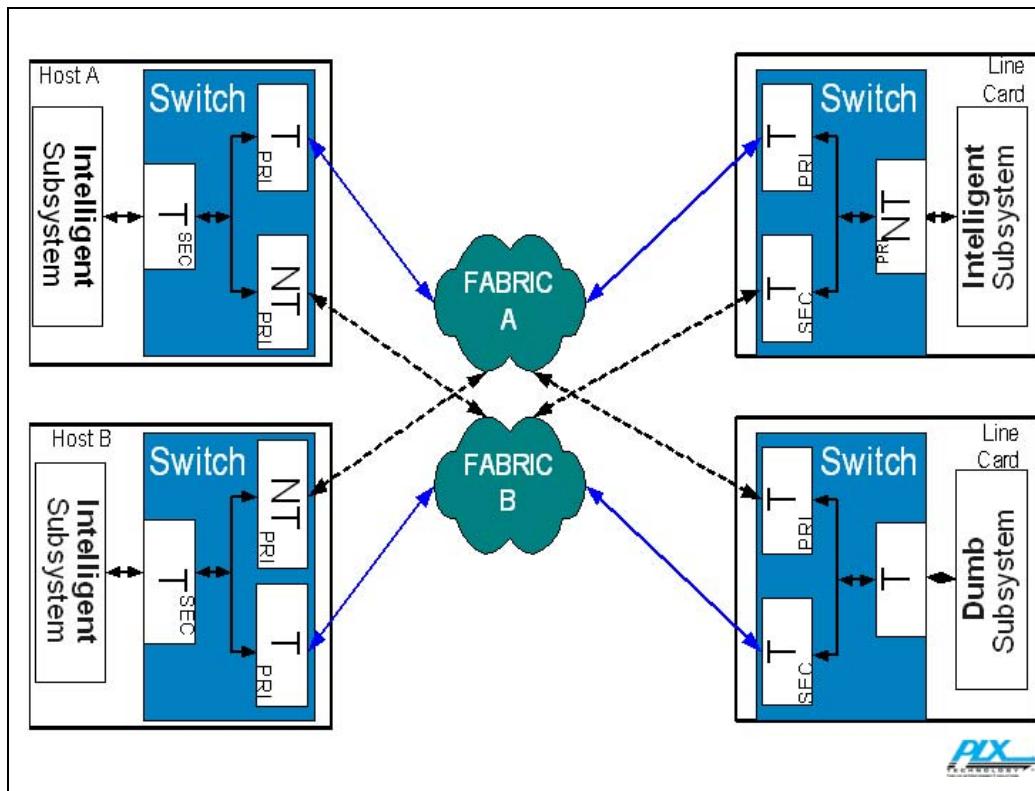
<sup>2</sup>The dual-host system model may be extended to a fully redundant dual star system by using additional switches to dual-port the hosts and line cards into a redundant fabric as shown in Figure C-0-7 on page 957. This is particularly attractive to vendors who employ chassis based systems for their flexibility, scalability and reliability.

Two host cards are shown. Host A is the primary host of Fabric A and the secondary host of Fabric B. Similarly, Host B is the primary host of Fabric B and the secondary host of Fabric A.

Each host is connected to the fabric it serves via a transparent bridge/switch port and to the fabric for which it provides only backup via a non-transparent bridge/switch port. These non-transparent ports are used for host-to-host communications and also support cross-domain peer-to-peer transfers where address maps do not allow a more direct connection.

## Chapter : Appendix C: Implementing Intelligent Adapt-

Figure 0-7: Dual-Star Fabric



## Summary

Through non-transparent bridging, PCI Express Base offers vendors the ability to integrate intelligent adapters and multi-host systems into their next generation designs. This appendix demonstrated how these features will be deployed using de-facto standard techniques adopted in the PCI environment and showed how they would be utilized for various applications. Because of this, we can expect this methodology to become the industry standard in the PCI Express paradigm.

# **PCI Express 3.0 Technology**

---

## **Address Translation**

This section provides an in-depth description of how systems that use non-transparent bridges communicate using address translation. We provide details about the mechanism by which systems determine not only the size of the memory allocated, but also about how memory pointers are employed. Implementations using both Direct Address Translation as well as Lookup Table Based Address Translation are discussed. By using the same standardized architectural implementation of non transparent bridging popularized in the PCI paradigm into the PCI Express environment, interconnect vendors can speed market adoption of PCI Express into markets requiring intelligent adapters, host failover and multihost capabilities.

The transparent bridge uses base and limit registers in I/O space, non-prefetchable memory space, and prefetchable memory space to map transactions in the downstream direction across the bridge. All downstream devices are required to be mapped in contiguous address regions such that a single aperture in each space is sufficient. Upstream mapping is done via inverse decoding relative to the same registers. A transparent bridge does not translate the addresses of forwarded transactions/packets.

The non-transparent bridges use the standard set of BARs in their Type 0 CSR header to define apertures into the memory space on the other side of the bridge. There are two sets of BARs: one on the Primary side and one on the Secondary. BARs define resource apertures that allow the forwarding of transactions to the opposite (other side) interface.

For each BAR bridge there exists a set of associated control and setup registers usually writable from the other side of the bridge. Each BAR has a “setup” register, which defines the size and type of its aperture, and an address translation register. Some bars also have a limit register that can be used to restrict its aperture’s size. These registers need to be programmed prior to allowing access from outside the local subsystem. This is typically done by software running on a local processor or by loading the registers from EEPROM.

In PCI Express, the Transaction ID fields of packets passing through these apertures are also translated to support Device ID routing. These Device IDs are used to route completions to non-posted requests and ID routed messages.

The transparent bridge forwards CSR transactions in the downstream direction according to the secondary and subordinate bus number registers, converting Type 1 CSRs to Type 0 CSRs as required. The non-transparent bridge accepts only those CSR transactions addressed to it and returns an unsupported request response to all others.

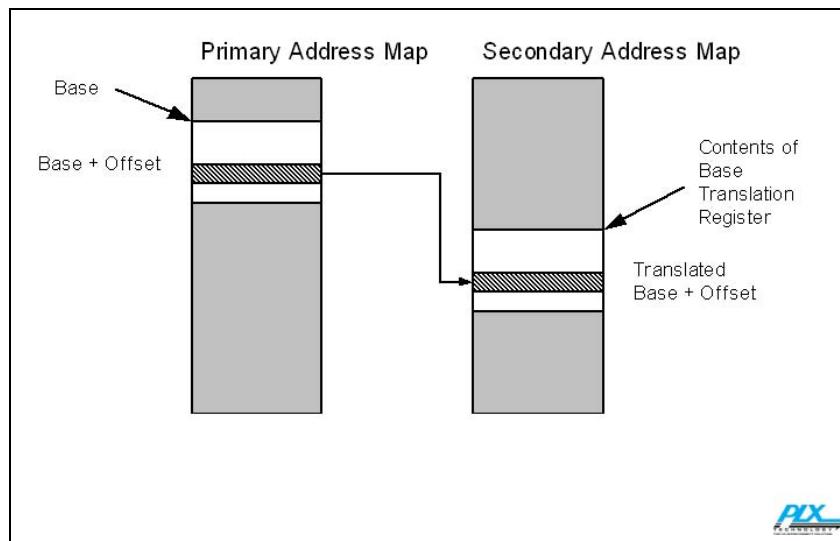
## **Chapter : Appendix C: Implementing Intelligent Adapt-**

---

### **Direct Address Translation**

The addresses of all upstream and downstream transactions are translated (except BARs accessing CSRs). With the exception of the cases in the following two sections, addresses that are forwarded from one interface to the other are translated by adding a Base Address to their offset within the BAR that they landed in as seen in Figure C-0-8 on page 959. The BAR Base Translation Registers are used to set up these base translations for the individual BARs.

*Figure 0-8: Direct Address Translation*



### **Lookup Table Based Address Translation**

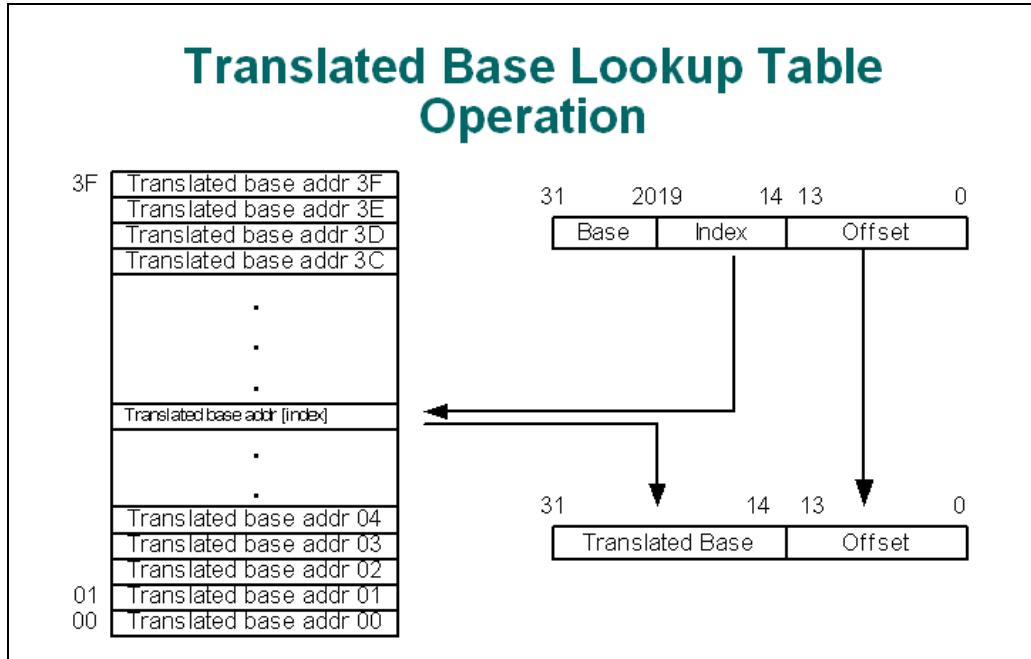
Following the de facto standard adopted by the PCI community, PCI Express should provide several BARs for the purposes of allocating resources. All BARs contain the memory allocation; however, in accordance with PCI industry conventions, BAR 0 contains the CSR information whereas BAR1 contains I/O information, BAR 2 and BAR 3 are utilized for Lookup Table Based Translation. BAR 4 and BAR 5 are utilized for Direct Address Translations.

On the secondary side, BAR3 uses a special lookup table based address translation for transactions that fall inside its window as seen in Figure C-0-9 on page 960. The lookup table provides more flexibility in secondary bus local addresses

# PCI Express 3.0 Technology

to primary bus addresses. The location of the index field with the address bus is programmable to adjust aperture size.

Figure 0-9: Lookup Table Based Translation

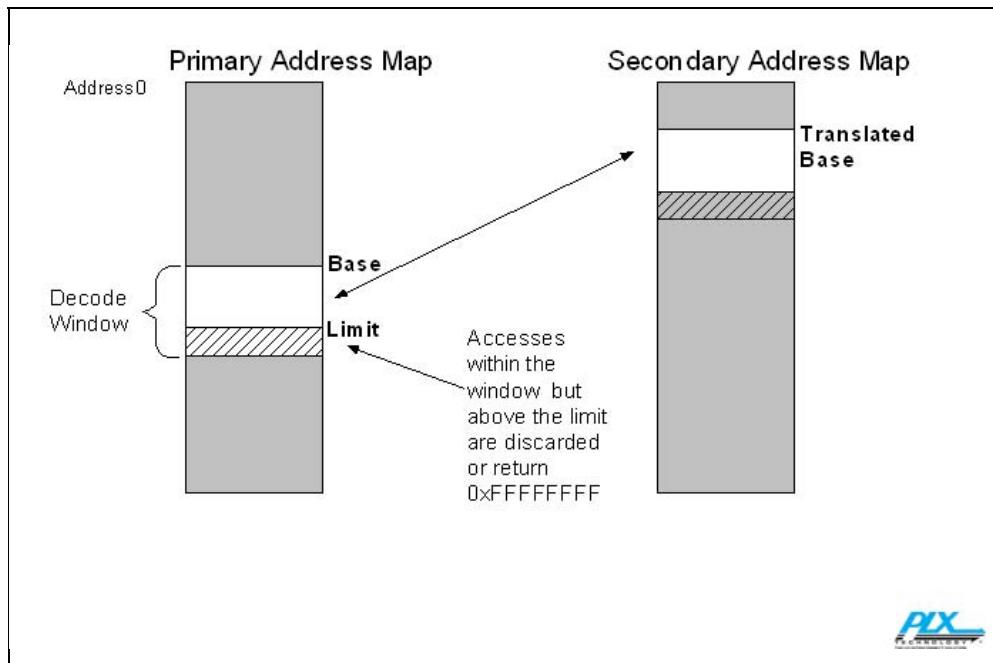


## Downstream BAR Limit Registers

The two downstream BARs on the primary side (BAR2/3 and BAR4/5) also have Limit registers, programmable from the local side, to further restrict the size of the window they expose, as seen in Figure C-0-10 on page 961. BARs can only be assigned memory resources in “power of two” granularity. The limit registers provide a means to obtain better granularity by “capping” the size of the BAR within the “power of two” granularity. Only transactions below the Limit registers are forwarded to the secondary bus. Transactions above the limit are discarded or return 0xFFFFFFFF, or a master abort equivalent packet, on reads.

## Chapter : Appendix C: Implementing Intelligent Adapt-

Figure 0-10: Use of Limit Register



### **Forwarding 64bit Address Memory Transactions**

Certain BARs can be configured to work in pairs to provide the base address and translation for transactions containing 64-bit addresses. Transactions that hit within these 64-bit BARs are forwarded using Direct Address Translation. As in the case of 32 bit transactions, when a memory transaction is forwarded from the primary to the secondary bus, the primary address can be mapped to another address in the secondary bus domain. The mapping is performed by substituting a new base address for the base of the original address.

A 64-bit BAR pair on the system side of the bridge is used to translate a window of 64-bit addresses in packets originated on the system side of the bridge down below 2<sup>32</sup> in local space.

## **PCI Express 3.0 Technology**

---

---

---

# *Appendix D:*

## *Locked Transactions*

---

### **Introduction**

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

---

### **Background**

PCI Express supports atomic or uninterrupted transaction sequences (usually described as an atomic read-modify-write sequence) for legacy devices only. Native PCIe devices don't support this at all and will return a Completion with UR (Unsupported Request) status if they receive a locked Request.

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

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

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

# PCI Express Technology

---

The devices required to support locked sequences are:

- The Root Complex.
- Any Switches in the path to a Legacy Device that may be the target of a locked transaction series.
- PCIe-to-PCI Bridge or PCIe-to-PCI-X Bridge.
- Any Legacy Device whose driver issues locked transactions to memory residing within the legacy device.

Locking in the PCI environment is achieved by the use of the LOCK# signal. The equivalent functionality in PCIe is accomplished by using a specific Request that emulates the LOCK# signal functionality.

---

## The PCI Express Lock Protocol

The only source of lock supported by PCI Express is the system processor acting through the Root Complex. A locked operation is performed between a Root Port and the Legacy Endpoint. In most systems, the legacy device is typically a PCI Express-to-PCI or PCI Express-to-PCI-X bridge. Only one locked sequence at a time is supported for a given hierarchical path.

Locked transactions are constrained to use only Traffic Class 0 and Virtual Channel 0. Transactions with other TC values that map to a VC other than zero are permitted to traverse the fabric without regard to the locked operation, but transactions that map to VC0 are affected by the lock rules described here.

---

## Lock Messages — The Virtual Lock Signal

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

- **Memory Read Lock Request** (MRdLk) — Originates a locked sequence. The first MRdLk transaction blocks other Requests in VC0 from reaching the target device. One or more of these locked read requests may be issued during the sequence.
- **Memory Read Lock Completion with Data** (CplDLk) — Returns data and confirms that the path to the target is locked. A successful read Completion that returns data for the first Memory Read Lock request results in the path between the Root Complex and the target device being locked. That is, transactions traversing the same path from other ports are blocked from reaching either the root port or the target port. Transactions being routed in buffers for VC1-VC7 are unaffected by the lock.

- **Memory Read Lock Completion without Data (CplLK)** — A Completion without a data payload indicates that the lock sequence cannot complete currently and the path remains unlocked.
- **Unlock Message** — An unlock message is issued by the Root Complex from the locked root port. This message unlocks the path between the root port and the target port.

---

### The Lock Protocol Sequence — an Example

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

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

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

### The Memory Read Lock Operation

Figure E-1 on page 967 illustrates the first step in the Locked transaction series (i.e., the initial memory read to obtain the semaphore):

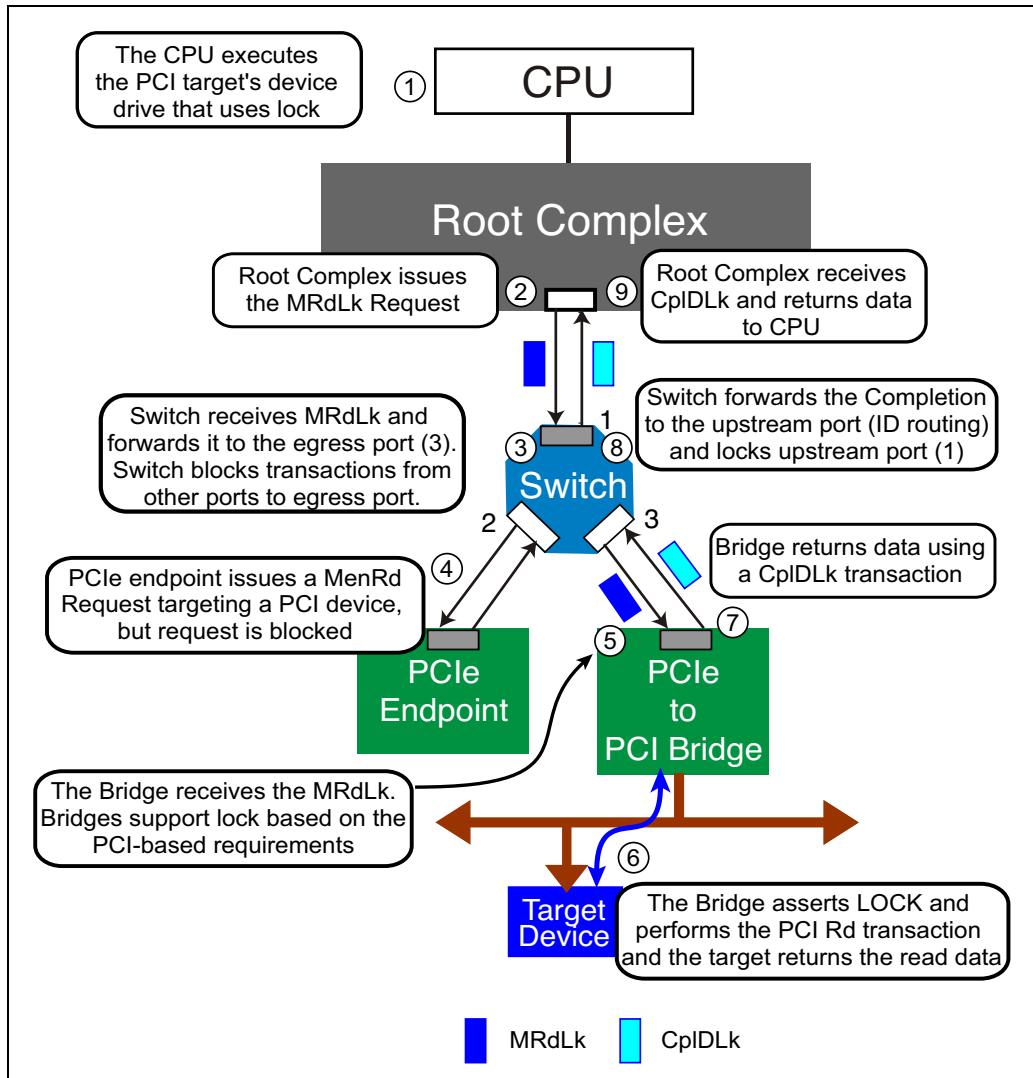
1. The CPU initiates the locked sequence (a Locked Memory Read) as a result of a driver executing a locked RMW instruction that targets a PCI target.
2. The Root Port issues a Memory Read Lock Request from port 2. The Root Complex is always the source of a locked sequence.
3. The Switch receives the lock request on its upstream port and forwards the request to the target egress port (3). The switch, upon forwarding the request to the egress port, must block all requests from ports other than the ingress port (1) from being sent from the egress port.
4. A subsequent peer-to-peer transfer from the illustrated PCI Express endpoint to the PCI bus (switch port 2 to switch port 3) would be blocked until the lock is cleared. Note that the lock is not yet established in the other direction. Transactions from the PCI Express endpoint could be sent to the Root Complex.

## PCI Express Technology

---

5. The Memory Read Lock Request is sent from the Switch's egress port to the PCI Express-to-PCI Bridge. This bridge will implement PCI lock semantics (See the MindShare book entitled *PCI System Architecture, Fourth Edition*, for details regarding PCI lock).
6. The bridge performs the Memory Read transaction on the PCI bus with the PCI LOCK# signal asserted. The target memory device returns the requested semaphore data to the bridge.
7. Read data is returned to the Bridge and is delivered back to the Switch via a Memory Read Lock Completion with Data (CplDLk).
8. The switch uses ID routing to return the packet upstream towards the host processor. When the CplDLk packet is forwarded to the upstream port of the Switch, it establishes a lock in the upstream direction to prevent traffic from other ports from being routed upstream. The PCI Express endpoint is completely blocked from sending any transaction to the Switch ports via the path of the locked operation. Note that transfers between Switch ports not involved in the locked operation would be permitted (not shown in this example).
9. Upon detecting the CplDLk packet, the Root Complex knows that the lock has been established along the path between it and the target device, and the completion data is sent to the CPU.

*Figure D-1: Lock Sequence Begins with Memory Read Lock Request*



### Read Data Modified and Written to Target and Lock Completes

The device driver receives the semaphore value, alters it, and then initiates a memory write to update the semaphore within the memory of the legacy PCI device. Figure E-2 on page 969 illustrates the write sequence followed by the

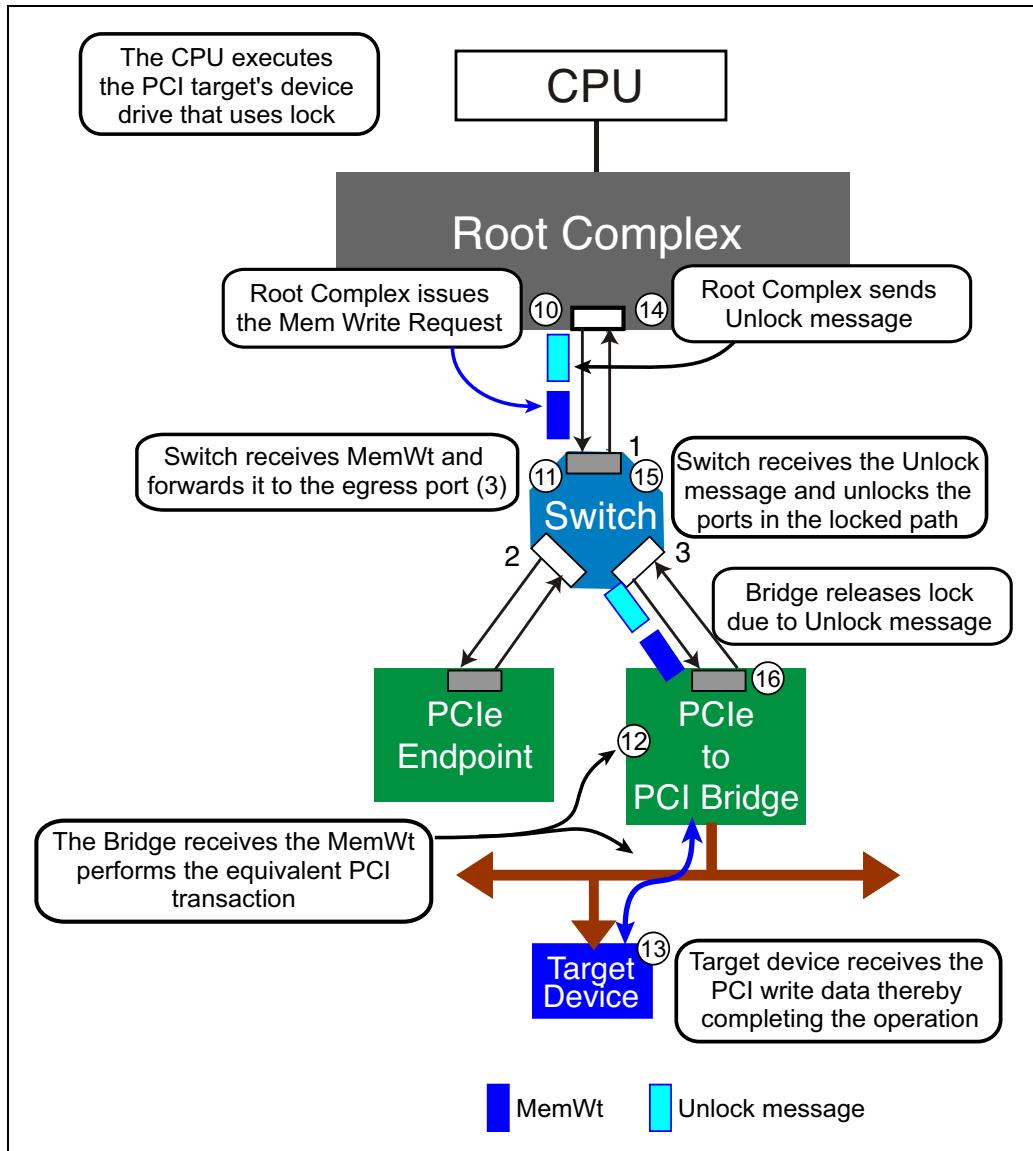
## **PCI Express Technology**

---

Root Complex's transmission of the Unlock message that releases the lock:

10. The Root Complex issues the Memory Write Request across the locked path to the target device.
11. The Switch forwards the transaction to the target egress port (3). The memory address of the Memory Write must be the same as the initial Memory Read request.
12. The bridge forwards the transaction to the PCI bus.
13. The target device receives the memory write data.
14. Once the Memory Write transaction is sent from the Root Complex, it sends an Unlock message to instruct the Switches and any PCI/PCI-X bridges in the locked path to release the lock. Note that the Root Complex presumes the operation has completed normally (because memory writes are posted and no Completion is returned to verify success).
15. The Switch receives the Unlock message, unlocks its ports and forwards the message to the egress port that was locked to notify any other Switches and/or bridges in the locked path that the lock must be cleared.
16. Upon detecting the Unlock message, the bridge must also release the lock on the PCI bus.

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



---

## Notification of an Unsuccessful Lock

A locked transaction series is aborted when the initial Memory Read Lock Request receives a Completion packet with no data (CplLk). This means that the locked sequence must terminate because no data was returned. This could result from an error associated with the memory read transaction, or perhaps the target device is busy and cannot respond at this time.

---

## Summary of Locking Rules

Following is a list of ordering rules that apply to the Root Complex, Switches, and Bridges.

---

## Rules Related To the Initiation and Propagation of Locked Transactions

- Locked Requests which are completed with a status other than Successful Completion do not establish lock.
- Regardless of the status of any of the Completions associated with a locked sequence, all locked sequences and attempted locked sequences must be terminated by the transmission of an Unlock Message.
- MRdLk, CplDLk and Unlock semantics are allowed only for the default Traffic Class (TC0).
- Only one locked transaction sequence attempt may be in progress at a given time within a single hierarchy domain.
- Any device which is not involved in the locked sequence must ignore the Unlock Message.

The initiation and propagation of a locked transaction sequence through the PCI Express fabric is performed as follows:

- A locked transaction sequence is started with a MRdLk Request:
  - Any successive reads associated with the locked transaction sequence must also use MRdLk Requests.
  - The Completions for any successful MRdLk Request use the CplDLk Completion type, or the CPIlk Completion type for unsuccessful Requests.

- If any read associated with a locked sequence is completed unsuccessfully, the Requester must assume that the atomicity of the lock is no longer assured, and that the path between the Requester and Completer is no longer locked.
- All writes associated with a locked sequence must use MWr Requests.
- The Unlock Message is used to indicate the end of a locked sequence. A Switch propagates Unlock Messages through the locked Egress Port.
- Upon receiving an Unlock Message, a legacy Endpoint or Bridge must unlock itself if it is in a locked state. If it is not locked, or if the Receiver is a PCI Express Endpoint or Bridge which does not support lock, the Unlock Message is ignored and discarded.

---

## Rules Related to Switches

Switches must detect transactions associated with locked sequences from other transactions to prevent other transactions from interfering with the lock and potentially causing deadlock. The following rules cover how this is done. Note that locked accesses are limited to TC0, which is always mapped to VC0.

- When a Switch propagates a MRdLk Request from an Ingress Port to the Egress Port, it must block all Requests which map to the default Virtual Channel (VC0) from being propagated to the Egress Port. If a subsequent MRdLk Request is received at this Ingress Port addressing a different Egress Port, the behavior of the Switch is undefined. Note that this sort of split-lock access is not supported by PCI Express and software must not cause such a locked access. System deadlock may result from such accesses.
- When the CplDLk for the first MRdLk Request is returned, if the Completion indicates a Successful Completion status, the Switch must block all Requests from all other Ports from being propagated to either of the Ports involved in the locked access, except for Requests which map to channels other than VC0 on the Egress Port.
- The two Ports involved in the locked sequence must remain blocked until the Switch receives the Unlock Message (at the Ingress Port which received the initial MRdLk Request)
  - The Unlock Message must be forwarded to the locked Egress Port.
  - The Unlock Message may be broadcast to all other Ports.
  - The Ingress Port is unblocked once the Unlock Message arrives, and the Egress Port(s) which were blocked are unblocked following the transmission of the Unlock Message out of the Egress Port(s). Ports that were not involved in the locked access are unaffected by the Unlock Message

## Rules Related To PCI Express/PCI Bridges

The requirements for PCI Express/PCI Bridges are similar to those for Switches, except that, because these Bridges only use TC0 and VC0, all other traffic is blocked during the locked access. Requirements on the PCI bus side are described in the MindShare book, *PCI System Architecture, Fourth Edition*.

## Rules Related To the Root Complex

A Root Complex is permitted to support locked transactions as a Requester. If locked transactions are supported, a Root Complex must follow the rules already described to perform a locked access. The mechanism(s) used by the Root Complex to interface to the host processor's FSB (Front-Side Bus) are outside the scope of the spec.

## Rules Related To Legacy Endpoints

Legacy Endpoints are permitted to support locked accesses, although their use is discouraged. If locked accesses are supported, legacy Endpoints must handle them as follows:

- The legacy Endpoint becomes locked when it transmits the first Completion for the first read request of the locked transaction series access with a Successful Completion status:
  - If the completion status is not Successful Completion, the legacy Endpoint does not become locked.
  - Once locked, the legacy Endpoint must remain locked until it receives the Unlock Message.
- While locked, a legacy Endpoint must not issue any Requests using Traffic Classes which map to the default Virtual Channel (VC0). Note that this requirement applies to all possible sources of Requests within the Endpoint, in the case where there is more than one possible source of Requests. Requests may be issued using TCs which map to VCs other than VC0.

## Rules Related To PCI Express Endpoints

Native PCI Express Endpoints do not support lock. A PCI Express Endpoint must treat a MRdLk Request as an Unsupported Request.

---

# *Glossary*

<b>Term</b>	<b>Definition</b>
128b/130b Encoding	This isn't encoding in the same sense as 8b/10b. Instead, the transmitter sends information in Blocks that consist of 16 raw bytes in a row, preceded by a 2-bit Sync field that indicates whether the Block is to be considered as a Data Block or an Ordered Set Block. This scheme was introduced with Gen3, primarily to allow the Link bandwidth to double without doubling the clock rate. It provides better bandwidth utilization but sacrifices some benefits that 8b/10b provided for receivers.
8b/10b Encoding	Encoding scheme developed many years ago that's used in many serial transports today. It was designed to help receivers recover the clock and data from the incoming signal, but it also reduces available bandwidth at the receiver by 20%. This scheme is used with the earlier versions of PCIe: Gen1 and Gen2.
ACK/NAK Protocol	The Acknowledge/Negative Acknowledge mechanism by which the Data Link Layer reports whether TLPs have experienced any errors during transmission. If so, a NAK is returned to the sender to request a replay of the failed TLPs. If not, an ACK is returned to indicate that one or more TLPs have arrived safely.
ACPI	Advanced Configuration and Power Interface. Specifies the various system and device power states.
ACS	Access Control Services.

# PCI Express Technology

Term	Definition
ARI	Alternative Routing-ID Interpretation; optional feature that allows Endpoints to have more Functions than the 8 allowed normally.
ASPM	Active State Power Management: When enabled, this allows hardware to make changes to the Link power state from L0 to L0s or L1 or both.
AtomicOps	Atomic Operations; three new Requests added with the 2.1 spec revision. These carry out multiple operations that are guaranteed to take place without interruption within the target device.
Bandwidth Management	Hardware-initiated changes to Link speed or width for the purpose of power conservation or reliability.
BAR	Base Address Register. Used by Functions to indicate the type and size of their local memory and IO space.
Beacon	Low-frequency in-band signal used by Devices whose main power has been shut off to signal that an event has occurred for which they need to have the power restored. This can be sent across the Link when the Link is in the L2 state.
BER	Bit Error Rate or Ratio; a measure of signal integrity based on the number of transmission bit errors seen within a time period
Bit Lock	The process of acquiring the transmitter's precise clock frequency at the receiver. This is done in the CDR logic and is one of the first steps in Link Training.
Block	The 130-bit unit sent by a Gen3 transmitter, made up of a 2-bit Sync Field followed by a group of 16 bytes.

## Glossary

---

Term	Definition
Block Lock	Finding the Block boundaries at the Receiver when using 128b/130b encoding so as to recognize incoming Blocks. The process involves three phases. First, search the incoming stream for an EIEOS (Electrical Idle Exit Ordered Set) and adjust the internal Block boundary to match it. Next, search for the SDS (Start Data Stream) Ordered Set. After that, the receiver is locked into the Block boundary.
Bridge	A Function that acts as the interface between two buses. Switches and the Root Complex will implement bridges on their Ports to enable packet routing, and a bridge can also be made to connect between different protocols, such as between PCIe and PCI.
Byte Striping	Spreading the output byte stream across all available Lanes. All available Lanes are used whenever sending bytes.
CC	Credits Consumed: Number of credits already used by the transmitter when calculating Flow Control.
CDR	Clock and Data Recovery logic used to recover the Transmitter clock from the incoming bit stream and then sample the bits to recognize patterns. For 8b/10b, that pattern, found in the COM, FTS, and EIEOS symbols, allows the logic to acquire Symbol Lock. For 128b/130b the EIEOS sequence is used to acquire Block Lock by going through the three phases of locking.
Character	Term used to describe the 8-bit values to be communicated between Link neighbors. For Gen1 and Gen2, these are a mix of ordinary data bytes (labeled as D characters) and special control values (labeled as K characters). For Gen3 there are no control characters because 8b/10b encoding is no longer used. In that case, the characters all appear as data bytes.

# PCI Express Technology

Term	Definition
CL	Credit Limit: Flow Control credits seen as available from the transmitter's perspective. Checked to verify whether enough credits are available to send a TLP.
Control Character	These are special characters (labeled as "K" characters) used in 8b/10b encoding that facilitate Link training and Ordered Sets. They are not used in Gen3, where there is no distinction between characters.
Correctable Errors	Errors that are corrected automatically by hardware and don't require software attention.
CR	Credits Required - this is the sum of CC and PTLP.
CRC	Cyclic Redundancy Code; added to TLPs and DLLPs to allow verifying error-free transmission. The name means that the patterns are cyclic in nature and are redundant (they don't add any extra information). The codes don't contain enough information to permit automatic error correction, but provide robust error detection.
Cut-Through Mode	Mechanism by which a Switch allows a TLP to pass through, forwarded from an ingress Port to an egress Port without storing it first to check for errors. This involves a risk, since the TLP may be found to have errors after part of it has already been forwarded to the egress Port. In that case, the end of the packet is modified in the Data Link Layer to have an LCRC value that is inverted from what it should be, and also modified at the Physical Layer to have an End Bad (EDB) framing symbol for 8b/10b encoding or an EDB token for 128b/130b encoding. The combination tells the receiver that the packet has been nullified and should be discarded without sending an ACK/NAK response.
Data Characters	Characters (labeled as "D" characters) that represent ordinary data and are distinguished from control characters in 8b/10b. For Gen3, there is no distinction between characters.

## Glossary

---

Term	Definition
Data Stream	The flow of data Blocks for Gen3 operation. The stream is entered by an SDS (Start of Data Stream Ordered Set) and exited with an EDS (End of Data Stream token). During a Data Stream, only data Blocks or the SOS are expected. When any other Ordered Sets are needed, the Data Stream must be exited and only re-entered when more data Blocks are ready to send. Starting a Data Stream is equivalent to entering the L0 Link state, since Ordered Sets are only sent while in other LTSSM states, like Recovery.
De-emphasis	The process of reducing the transmitter voltage for repeated bits in a stream. This has the effect of de-emphasizing the low-frequency components of the signal that are known to cause trouble in a transmission medium and thus improves the signal integrity at the receiver.
Digest	Another name for the ECRC (End-to-End CRC) value that can optionally be appended to a TLP when it's created in the Transaction Layer.
DLCMSM	Data Link Control and Management State Machine; manages the Link Layer training process (which is primarily Flow Control initialization).
DLLP	Data Link Layer Packet. These are created in the Data Link Layer and are forwarded to the Physical Layer but are not seen by the Transaction Layer.
DPA	Dynamic Power Allocation; a new set of configuration registers with the 2.1 spec revision that defines 32 power substates under the D0 device power state, making it easier for software to control device power options.
DSP (Downstream Port)	Port that faces downstream, like a Root Port or a Switch Downstream Port. This distinction is meaningful in the LTSSM because the Ports have assigned roles during some states.

# PCI Express Technology

---

Term	Definition
ECRC	End-to-End CRC value, optionally appended to a TLP when it's created in the Transaction Layer. This enables a receiver to verify reliable packet transport from source to destination, regardless of how many Links were crossed to get there.
Egress Port	Port that has outgoing traffic.
Elastic Buffer	Part of the CDR logic, this buffer enables the receiver to compensate for the difference between the transmitter and receiver clocks.
EMI	Electro-Magnetic Interference: the emitted electrical noise from a system. For PCIe, both SSC and scrambling are used to attack it.
Endpoint	PCIe Function that is at the bottom of the PCI Inverted-Tree structure.
Enumeration	The process of system discovery in which software reads all of the expected configuration locations to learn which PCI-configurable Functions are visible and thus present in the system.
Equalization	The process of adjusting Tx and Rx values to compensate for actual or expected signal distortion through the transmission media. For Gen1 and Gen2, this takes the form of Tx De-emphasis. For Gen3, an active evaluation process is introduced to test the signaling environment and adjust the Tx settings accordingly, and optional Rx equalization is mentioned.
Flow Control	Mechanism by which transmitters avoid the risk of having packets rejected at a receiver due to lack of buffer space. The receiver sends periodic updates about available buffer space and the transmitter verifies that enough is available before attempting to send a packet.
FLR	Function-Level Reset

## Glossary

---

Term	Definition
Framing Symbols	The “start” and “end” control characters used in 8b/10b encoding that indicate the boundaries of a TLP or DLLP.
Gen1	Generation 1, meaning designs created to be compliant with the 1.x version of the PCIe spec.
Gen1, Gen2, Gen3	Abbreviations for the revisions of the PCIe spec. Gen1 = rev 1.x, Gen2 = rev 2.x, and Gen3 = rev 3.0
Gen2	Generation 2, meaning designs created to be compliant with the 2.x version of the PCIe spec.
Gen3	Generation 3, meaning designs created to be compliant with the 3.x version of the PCIe spec.
IDO	ID-based Ordering; when enabled, this allows TLPs from different Requesters to be forwarded out of order with respect to each other. The goal is to improve latency and performance.
Implicit Routing	TLPs whose routing is understood without reference to an address or ID. Only Message requests have the option to use this type of routing.
Ingress Port	Port that has incoming traffic.
ISI	Inter-Symbol Interference; the effect on one bit time that is caused by the recent bits that preceded it.
Lane	The two differential pairs that allow a transmit and receive path of one bit between two Ports. A Link can consist of just one Lane or it may contain as many as 32 Lanes.
Lane-to-Lane Skew	Difference in arrival times for bits on different Lanes. Receivers are required to detect this and correct it internally.
Legacy Endpoint	An Endpoint that carries any of three legacy items forward: support for IO transactions, support for local 32-bit-only prefetchable memory space, or support for the locked transactions.

# PCI Express Technology

---

Term	Definition
LFSR	Linear-Feedback Shift Register; creates a pseudo-random pattern used to facilitate scrambling.
Link	Interface between two Ports, made up of one or more Lanes.
LTR	Latency-Tolerance Reporting; mechanism that allows devices to report to the system how quickly they need to get service when they send a Request. Longer latencies afford more power management options to the system.
LTSSM	Link Training and Status State Machine; manages the training process for the Physical Layer.
Non-posted Request	A Request that expects to receive a Completion in response. For example, any read request would be non-posted.
Non-prefetchable Memory	Memory that exhibits side effects when read. For example, a status register that automatically self-clears when read. Such data is not safe to prefetch since, if the requester never requested the data and it was discarded, it would be lost to the system. This was an important distinction for PCI bridges, which had to guess about the data size on reads. If they knew it was safe to speculatively read ahead in the memory space, they could guess a larger number and achieve better efficiency. The distinction is much less interesting for PCIe, since the exact byte count for a transfer is included in the TLP, but maintaining it allows backward compatibility.
Nullified Packet	When a transmitter recognizes that a packet has an error and should not have been sent, the packet can be “nullified”, meaning it should be discarded and the receiver should behave as if it had never been sent. This problem can arise when using “cut-through” operation on a Switch.

## Glossary

---

Term	Definition
OBFF	Optimized Buffer Flush and Fill; mechanism that allows the system to tell devices about the best times to initiate traffic. If devices send requests during optimal times and not during other times system power management will be improved.
Ordered Sets	Groups of Symbols sent as Physical Layer communication for Lane management. These often consist of just control characters for 8b/10b encoding. They are created in the Physical Layer of the sender and consumed in the Physical Layer of the receiver without being visible to the other layers at all.
PCI	Peripheral Component Interface. Designed to replace earlier bus designs used in PCs, such as ISA.
PCI-X	PCI eXtended. Designed to correct the shortcomings of PCI and enable higher speeds.
PME	Power Management Event; message from a device indicating that power-related service is needed.
Poisoned TLP	Packet whose data payload was known to be bad when it was created. Sending the packet with bad data can be helpful as an aid to diagnosing the problem and determining a solution for it.
Polarity Inversion	The receiver's signal polarity is permitted to be connected backwards to support cases when doing so would simplify board layout. The receiver is required to detect this condition and internally invert the signal to correct it during Link Training.
Port	Input/output interface to a PCIe Link.
Posted Request	A Request packet for which no completion is expected. There are only two such requests defined by the spec: Memory Writes and Messages.

# PCI Express Technology

---

Term	Definition
Prefetchable Memory	Memory that has no side-effects as a result of being read. That property makes it safe to prefetch since, if it's discarded by the intermediate buffer, it can always be read again later if needed. This was an important distinction for PCI bridges, which had to guess about the data size on reads. Prefetchable space allowed speculatively reading more data and gave a chance for better efficiency. The distinction is much less interesting for PCIe, since the exact byte count for a transfer is included in the TLP, but maintaining it allows backward compatibility.
PTLP	Pending TLP - Flow Control credits needed to send the current TLP.
QoS	Quality of Service; the ability of the PCIe topology to assign different priorities for different packets. This could just mean giving preference to packets at arbitration points, but in more complex systems, it allows making bandwidth and latency guarantees for packets.
Requester ID	The configuration address of the Requester for a transaction, meaning the BDF (Bus, Device, and Function number) that corresponds to it. This will be used by the Completer as the return address for the resulting completion packet.
Root Complex	The components that act as the interface between the CPU cores in the system and the PCIe topology. This can consist of one or more chips and may be simple or complex. From the PCIe perspective, it serves as the root of the inverted tree structure that backward-compatibility with PCI demands.
Run Length	The number of consecutive ones or zeros in a row. For 8b/10b encoding the run length is limited to 5 bits. For 128b/130b, there is no defined limit, but the modified scrambling scheme it uses is intended to compensate for that.

## Glossary

---

Term	Definition
Scrambling	The process of randomizing the output bit stream to avoid repeated patterns on the Link and thus reduce EMI. Scrambling can be turned off for Gen1 and Gen2 to allow specifying patterns on the Link, but it cannot be turned off for Gen3 because it does other work at that speed and the Link is not expected to be able to work reliably without it.
SOS	Skip Ordered Set - used to compensate for the slight frequency difference between Tx and Rx.
SSC	Spread-Spectrum Clocking. This is a method of reducing EMI in a system by allowing the clock frequency to vary back and forth across an allowed range. This spreads the emitted energy across a wider range of frequencies and thus avoids the problem of having too much EMI energy concentrated in one particular frequency.
Sticky Bits	Status bits whose value survives a reset. This characteristic is useful for maintaining status information when errors are detected by a Function downstream of a Link that is no longer operating correctly. The failed Link must be reset to gain access to the downstream Functions, and the error status information in its registers must survive that reset to be available to software.
Switch	A device containing multiple Downstream Ports and one Upstream Port that is able to route traffic between its Ports.
Symbol	Encoded unit sent across the Link. For 8b/10b these are the 10-bit values that result from encoding, while for 128b/130b they're 8-bit values.
Symbol Lock	Finding the Symbol boundaries at the Receiver when using 8b/10b encoding so as to recognize incoming Symbols and thus the contents of packets.
Symbol time	The time it takes to send one symbol across the Link - 4ns for Gen1, 2ns for Gen2, and 1ns for Gen3.

# PCI Express Technology

<b>Term</b>	<b>Definition</b>
TLP	Transaction Layer Packet. These are created in the Transaction Layer and passed through the other layers.
Token	Identifier of the type of information being delivered during a Data Stream when operating at Gen3 speed.
TPH	TLP Processing Hints; these help system routing agents make choices to improve latency and traffic congestion.
UI	Unit Interval; the time it takes to send one bit across the Link - 0.4ns for Gen1, 0.2ns for Gen2, 0.125ns for Gen3
Uncorrectable Errors	Errors that can't be corrected by hardware and thus will ordinarily require software attention to resolve. These are divided into Fatal errors - those that render further Link operation unreliable, and Non-fatal errors - those that do not affect the Link operation in spite of the problem that was detected.
USP	Upstream Port, meaning a Port that faces upstream, as for an Endpoint or a Switch Upstream Port. This distinction is meaningful in the LTSSM because the Ports have assigned roles during Configuration and Recovery.

## Glossary

---

Term	Definition
Variables	<p>A number of flags are used to communicate events and status between hardware layers. These are specific to state transitions in the hardware and are not usually visible to software. Some examples:</p> <ul style="list-style-type: none"><li>– LinkUp - Indication from the Physical Layer to the Data Link Layer that training has completed and the Physical Layer is now operational.</li><li>– idle_to_rlock_transitioned - This counter tracks the number of times the LTSSM has transitioned from Configuration.Idle to the Recovery.RcvrLock state. Any time the process of recognizing TS2s to leave Configuration doesn't work, the LTSSM transitions to Recovery to take appropriate steps. If it still doesn't work after 256 passes through Recovery (counter reaches FFh), then it goes back to Detect to start over. It may be that some Lanes are not working.</li></ul>
WAKE#	Side-band pin used to signal to the system that the power should be restored. It's used instead of the Beacon in systems where power conservation is an important consideration.

## **PCI Express Technology**

---

---

## **Numerics**

128b/130b 43  
128b/130b Encoding 973  
1x Packet Format 374, 375  
3DW Header 152  
3-Tap Transmitter Equalization 585  
4DW Headers 152  
4x Packet Format 374  
8.0 GT/s 410  
8b/10b 42  
8b/10b Decoder 367  
8b/10b Encoder 366  
8b/10b Encoding 973

## **A**

AC Coupling 468  
ACK 318  
Ack 311  
ACK DLLP 75, 312  
ACK/NAK DLLP 312  
ACK/NAK Latency 328  
ACK/NAK Protocol 318, 320, 329, 973  
Ack/Nak Protocol 74  
ACKD\_SEQ Count 323  
ACKNAK\_Latency\_Timer 328, 343  
ACPI 711, 973  
ACPI Driver 706  
ACPI Machine Language 712  
ACPI Source Language 712  
ACPI spec 705  
ACPI tables 712  
ACS 973  
Active State Power Management 405, 735  
Address Routing 158  
Address Space 121  
Address Translation 958, 959  
Advanced Correctable Error Reporting 690  
Advanced Correctable Error Status 689  
Advanced Correctable Errors 688  
Advanced Error Reporting 685  
Advanced Source ID Register 697  
Advanced Uncorrectable Error Handling 691  
Advanced Uncorrectable Error Status 691  
Aggregate Bandwidth 408  
Alternative Routing-ID Interpretation 909  
AML 712  
AML token interpreter 712  
Arbitration 20, 270  
Arbor 117  
Architecture Overview 39  
ARI 909, 974  
ASL 712  
ASPM 735, 742, 910, 974  
ASPM Exit Latency 756, 757  
Assert\_INTx messages 806  
Async Notice of Slot Status Change 876

AtomicOp 150  
AtomicOps 897, 974  
Attention Button 854, 862  
Attention Indicator 854, 859  
Aux\_Current field 726

## **B**

Bandwidth 42  
Bandwidth Congestion 281  
Bandwidth Management 974  
BAR 126, 960, 974  
Base Address Registers 126  
Base and Limit Registers 136  
BDF 85  
Beacon 483, 772, 974  
BER 974  
BIOS 712, 853  
Bit Lock 78, 395, 507, 742, 974  
Bit Tracer 929  
Block 974  
Block Alignment 435  
Block Encoding 410  
Block Lock 507, 975  
Boost 476  
Bridge 975  
Bus 85  
Bus Master 20  
Bus Number register 93  
Byte Count Modified 201  
Byte Enables 181  
Byte Striping 371, 372, 373, 975  
byte striping 371  
Byte Striping logic 365  
Byte Un-Striping 402

## **C**

Capabilities List bit 818  
Capabilities Pointer register 713  
Capability ID 713, 814  
Capability Structures 88  
Card Connector Power Switching Logic 854  
Card Insertion 855  
Card Insertion Procedure 857  
Card Present 854  
Card Removal 855  
Card Removal Procedure 856  
Card Reset Logic 854  
CC 975  
CDR 435, 437, 975  
Character 79, 366, 975  
CL 976  
Class driver 706  
Clock Requirements 452  
Code Violation 400  
Coefficients 584  
Cold Reset 834

COM 386  
 Common-Mode Noise Rejection 452  
 Completer 33  
 Completer Abort 664  
 Completion Packet 197  
 Completion Status 200  
 Completion Time-out 665  
 Completion TLP 184  
 Completions 196, 218  
 Compliance Pattern 537  
 Compliance Pattern - 8b/10b 529  
 Configuration 85  
 Configuration Address Port 92, 93  
 Configuration Address Space 88  
 Configuration Cycle Generation 26  
 Configuration Data Port 92, 93  
 Configuration Headers 50  
 Configuration Read 151  
 Configuration Read Access 104  
 Configuration Register Space 27, 89  
 Configuration Registers 90  
 Configuration Request Packet 193  
 Configuration Requests 99, 192  
 Configuration Space 122  
 Configuration State 520, 540  
 Configuration Status Register 676  
 Configuration Status register 713  
 Configuration Transactions 91  
 Configuration Write 151  
 Configuration.Complete 562  
 Configuration.Idle 566  
 Configuration.Lanenum.Accept 560  
 Configuration.Lanenum.Wait 559  
 Configuration.Linkwidth.Accept 558  
 Configuration.Linkwidth.Start 553  
 Congestion Avoidance 897  
 Continuous-Time Linear Equalization 493  
 Control Character 976  
 Control Character Encoding 386  
 Control Method 712  
 Conventional Reset 834  
 Correctable Errors 651, 976  
 CR 976  
 CRC 976  
 CRD 383  
 Credit Allocated Count 229  
 Credit Limit counter 228  
 CREDIT\_ALLOCATED 229  
 Credits Consumed counter 228  
 Credits Received Counter 229  
 CREDITS RECEIVED 229  
 CTLE 493, 494  
 Current Running Disparity 383  
 Cursor Coefficient 584  
 Cut-Through 354  
 Cut-Through Mode 976

**D**  
 D0 709, 710, 714, 734  
 D0 Active 714  
 D0 Uninitialized 714  
 D1 709, 710, 716, 734  
 D1\_Support bit 725  
 D2 709, 710, 717, 734  
 D2\_Support bit 725  
 D3 709, 710, 719  
 D3cold 721, 734  
 D3hot 719, 734  
 Data Characters 976  
 Data Link Layer 55, 72  
 Data Link Layer Packet 72  
 Data Link Layer Packet Format 310  
 Data Link Layer Packets 73  
 Data Poisoning 660  
 Data Register 731  
 Data Stream 977  
 Data\_Scale field 729  
 Data\_Select field 729  
 DC Common Mode 462  
 DC Common Mode Voltage 466  
 DC Common-Mode Voltage 467  
 Deadlock Avoidance 303  
 Deassert\_INTx messages 806  
 Debugging PCIe Traffic 917  
 Decision Feedback Equalization 495  
 De-emphasis 450, 468, 469, 471, 476, 977  
 De-Scrambler 367  
 Deserializer 395  
 De-Skew 399  
 Detect State 519, 522  
 Detect.Active 524  
 Detect.Quiet 523  
 Device 85  
 Device Capabilities 2 Register 899  
 Device Capabilities Register 873  
 Device Context 709  
 Device Core 59  
 Device core 55  
 Device Driver 706  
 device driver 853  
 Device Layers 54  
 Device PM States 713  
 device PM states 709  
 Device Status Register 681  
 Device-Specific Initialization (DSI) bit 727  
 DFE 493, 495, 497  
 Differential Driver 389  
 Differential Receiver 393, 435, 451  
 Differential Signaling 463  
 Differential Signals 44  
 Differential Transmitter 451  
 Digest 180, 977  
 Direct Address Translation 949

Disable State 521, 613  
Discrete Time Linear Equalizer 493  
Discrete-Time Linear Equalizer 494  
Disparity 383  
Disparity Error Detection 400  
DLCMSM 977  
DLE 493, 494  
DLL 437  
DLLP 73, 170, 238, 308, 311, 977  
DLLP Elements 307  
DMA 937  
DPA 910, 977  
Driver Characteristics 489  
DSI bit 727  
DSP 977  
D-State Transitions 722  
Dual Simplex 363  
Dual-Simplex 40  
Dual-Star Fabric 957  
Dynamic Bandwidth Changes 618  
Dynamic Link Speed Changes 619  
Dynamic Link Width Changes 629  
Dynamic Power Allocation 910

## E

ECRC 63, 180, 978  
ECRC Generation and Checking 657  
EDB 373, 387  
Egress Port 978  
EIE 387  
EIEOS 389, 739, 740  
EIOS 388, 737  
Elastic Buffer 366, 435, 978  
Electrical Idle 388, 736, 738, 741  
Electrical Idle Exit Ordered Set 389  
Electrical Idle Ordered Set 388  
EMI 77, 978  
Encoding 410  
END 373, 387  
Endpoint 978  
End-to-End CRC 180  
Enhanced Configuration Access  
  Mechanism 96  
Enumeration 51, 104, 978  
Equalization 474, 978  
Equalization - Phase 0 578  
Equalization - Phase 1 581  
Equalization - Phase 2 583  
Equalization - Phase 3 586  
Equalization Control 513  
Equalization Control Registers 579  
Equalizer 475  
Equalizer Coefficients 479  
Error Classifications 651  
Error Handling 282, 699  
Error Isolation 937  
Error Messages 209, 668

ESD 459  
ESD standards 448  
Exerciser Card 931  
Extended Configuration Space 89  
Eye Diagram 486

## F

Failover 942, 944, 952  
FC Initialization 223  
FC Initialization Sequence 223  
FC\_Init1 224  
FC\_Init2 225  
FC\_Update 238  
First DW Byte Enables 178, 181  
Flow Control 72, 76, 215, 217, 299,  
  928, 978  
Flow Control Buffer 217, 229  
Flow Control Buffers 217  
Flow Control Credits 216, 219  
Flow Control Elements 227, 231  
Flow Control Initialization 227, 230, 237  
Flow Control Packet 239  
Flow Control Packet Format 314  
Flow Control Update Frequency 239  
Flow Control Updates 237  
FLR 842, 844, 845, 978  
Flying Lead Probe 924  
Format Field 179  
Framing Symbols 171, 979  
FTS 387  
FTS Ordered Set 388  
FTSOS 388  
Function 85  
Function Level Reset 842, 843  
Function PM State Transitions 722  
Function State Transition Delays 724  
Fundamental Reset 834

## G

Gen1 43, 77, 979  
Gen2 43, 77, 979  
Gen3 44, 77, 407, 979  
Gen3 products 936

## H

handler 712  
Hardware Based Fixed Arbitration 256  
Hardware Fixed VC Arbitration 257  
Hardware-Fixed Port Arbitration 265  
Header Type 0 29  
Header Type 1 28  
Header Type/Format Field 178  
High Speed Signaling 451  
host/PCI bridge 94  
Hot Plug 847, 852

Hot Plug Controller 863  
Hot Plug Elements 852  
Hot Plug Messages 211  
Hot Reset 839  
Hot Reset State 521, 612  
Hot-Plug 116, 853  
Hot-Plug Controller 853, 864  
hot-plug primitives 874  
Hot-Plug Service 852  
Hot-Plug System Driver 852  
HPC Applications 940  
Hub Link 32

## I

ID Based Ordering 301  
ID Routing 155  
ID-based Ordering 301, 909, 979  
IDL 387  
IDO 301, 302, 909, 979  
IEEE 1394 Bus Driver 711  
Ignored Messages 211  
Implicit Routing 148, 979  
In-band Reset 837  
Infinite Credits 221  
Infinite Flow Control Credits 219  
Ingress Port 979  
InitFC1-Cpl 312  
InitFC1-NP 311  
InitFC1-P DLLP 311  
InitFC2-Cpl 312  
InitFC2-NP 312  
InitFC2-P 312  
Intelligent Adapters 943, 944, 951  
Internal Error Reporting 911  
Interrupt Disable 803  
Interrupt Latency 829  
interrupt latency 829  
Interrupt Line Register 802  
Interrupt Pin Register 801  
Interrupt Status 804  
Inter-symbol Interference 469  
INTx Interrupt Messages 206  
INTx Interrupt Signaling 206  
INTx Message Format 807  
INTx# Pins 800  
INTx# Signaling 803  
IO 126  
IO Address Spaces 122  
IO Range 141  
IO Read 151  
IO Requests 184  
IO Virtualization 937  
IO Write 151  
ISI 979  
Isochronous Packets 279  
Isochronous Support 272  
Isochronous Transaction Support 272

## J

Jitter 485, 487

## L

L0 State 500, 520, 568  
L0s 744  
L0s Receiver State Machine 605  
L0s State 520, 603, 744  
L0s Transmitter State Machine 603  
L1 ASPM 736, 747  
L1 ASPM Negotiation 748  
L1 ASPM State 747  
L1 State 520, 607, 760  
L2 State 521, 609, 767  
L2/L3 Ready 767  
L2/L3 Ready state 763, 764  
Lane 40, 78, 365, 979  
Lane # 511  
Lane Number Negotiation 543, 547  
Lane Reversal 507  
Lane-Level Encoding 410  
Lane-to-Lane de-skew 78  
Lane-to-Lane Skew 979  
Last DW Byte Enables 178, 181  
Latency Tolerance Reporting 910  
LCRC 63, 325, 329  
LeCroy 922, 923, 933  
LeCroy Tools 917  
Legacy Endpoint 816, 979  
Legacy Endpoints 972  
LFSR 980  
Link 40, 980  
Link # 511  
Link Capabilities 2 Register 640  
Link Capability Register 743  
Link Configuration - Failed Lane 549  
Link Control 841  
Link Data Rate 509  
Link data rate 78  
Link Equalization 577  
Link Errors 683  
Link Flow Control-Related Errors 666  
Link Number Negotiation 542, 546  
Link Power Management 733  
Link Status Register 641  
Link Training and Initialization 78  
Link Training and Status State  
Machine (LTSSM) 518  
Link Upconfigure Capability 512  
Link Width 507  
Link width 78  
Link Width Change 570  
Link Width Change Example 630  
Lock 964  
Locked Reads 66  
Locked Transaction 209

Locked Transactions 963  
Logic Analyzer 921  
Logical Idle Sequence 370  
Loopback Master 615  
Loopback Slave 616  
Loopback State 521, 613  
Loopback.Active 617  
Loopback.Entry 614  
Loopback.Exit 618  
Low-priority VC Arbitration 255  
LTR 784, 910, 980  
LTR Messages 786  
LTR Registers 784  
LTSSM 507, 518, 839, 927, 980

## M

Malformed TLP 666  
Memory Address Space 122  
Memory Read 150  
Memory Read Lock 150  
Memory Request Packet 188  
Memory Requests 188  
Memory Write 150  
Memory Writes 69  
Message 151  
Message Address Register 816  
Message Address register 816, 818  
Message Control Register 814  
Message Control register 814, 818  
Message Data register 817, 818  
Message Request Packet 203  
Message Requests 70, 203  
Message Writes 70  
Messages 148  
Mid-Bus Probe 923  
MindShare Arbor 117  
Miniport Driver 706  
MMIO 123  
Modified Compliance Pattern 537  
Modified Compliance Pattern - 8b/10b 532  
MR-IOV 937, 939  
MSI Capability Register 812  
MSI Configuration 817  
Multicast 893, 896  
Multicast Capabilities 163  
Multicast Capability Registers 889  
Multi-casting 888  
Multi-Function Arbitration 272  
Multi-Host System 96  
Multi-Host Systems 943  
Multiple Message Capable field 818  
Multiple Messages 820  
Multi-Root 938  
Multi-Root Enumeration 114  
Multi-Root System 97, 116

## N

N\_FTS 511  
Nak 311  
NAK\_SCHEDULED Flag 327  
namespace 712  
Native PCI Express Endpoints 972  
NEXT\_RCV\_SEQ 313, 326, 341  
Noise 485  
Non-Posted 150  
non-posted 60  
Non-posted Request 980  
Non-Posted Transactions 65, 218  
Non-prefetchable 123  
Non-prefetchable Memory 980  
Non-Prefetchable Range 139  
North Bridge 21  
NP-MMIO 126, 139  
NT bridging 936  
Nullified Packet 388, 689, 980

## O

OBFF 776, 910, 981  
OBFF Messages 213  
OnNow Design Initiative 707  
Optimized Buffer Flush and Fill 776,  
910, 981  
Optimized Buffer Flush and Fill Messages 213  
Ordered Sets 981  
Ordered-Sets 370  
Ordering Rules 287  
Ordering Rules Table 288, 289  
Ordering Table 914  
Oscilloscope 919

## P

Packet Format 151  
Packet Generation 937  
Packet-Based Protocol 169  
Packet-based Protocol 46  
PAD 386  
Pause command 853, 874  
Pausing a Driver 874  
PCI 981  
PCI Bus Driver 706, 707, 711  
PCI Bus PM Interface Specification 705  
PCI Express 39  
PCI PM 705  
PCI power management 647, 703, 793  
PCI Transaction Model 18  
PCI-Based System 11  
PCI-Compatible Error Reporting 674  
PCIe System 53, 54  
PCI-X 981  
PERST# 835, 849  
PETracer 918, 924

PETrainer 932  
Physical Layer 55, 76  
Physical Layer Electrical 449  
PLL 435, 437  
PLX Technology 935, 943  
PM Capabilities (PMC) Register 724  
PM Capability Registers 713  
PM Control/Status (PMCSR) Register 727  
PM Registers 724, 732  
PM\_Active\_State\_Request\_L1 311  
PM\_Enter\_L1 DLLP 311  
PM\_Enter\_L23 311  
PM\_Request\_Ack 311  
PMC Register 724  
PMCSR 727, 728  
PMCSR Register 727  
PME 981  
PME Clock bit 727  
PME Context 710  
PME Generation 768  
PME Message 769  
PME\_En bit 730  
PME\_Status bit 728  
PME\_Support field 725  
P-MMIO 126, 137  
Poisoned TLP 981  
Polarity Inversion 78, 508, 981  
Polling State 519, 525  
Polling.Active 526  
Polling.Compliance 529  
Polling.Configuration 527  
Port 981  
Port Arbitration 261, 265  
Port Arbitration Table 267  
Port Arbitration Tables 263  
Post-Cursor Coefficient 584  
Posted 150  
posted 60  
Posted Request 981  
Posted Transactions 218  
Posted Writes 69  
Post-Silicon 931  
Post-Silicon Debug 919  
Power Budget Capabilities Register 883  
Power Budget Capability Registers 884  
Power Budget Registers 878  
Power Budgeting 847, 876  
Power Indicator 854, 860  
Power Management 76, 703, 711  
power management 647, 703, 793  
Power Management DLLP 313  
Power Management DLLP Packet 313  
Power Management Message 208  
Power Management Messages 208  
Power Management Policy Owner 711  
power management register set 713, 724  
Power Management States 500  
PowerState field 730

Pre-Cursor Coefficient 584  
Prefetchable 123  
Prefetchable Memory 982  
Prefetchable Range 137  
Presets 478  
Pre-shoot 476  
Pre-Silicon 931  
Pre-silicon Debugging 918  
Primitives 852  
Primitives, hot-plug 852, 874  
Producer/Consumer Model 290  
Producer/Consumer model 290  
Protocol Analyzer 920  
PTC card 932  
PTLP 982

**Q**

QoS 70, 245, 272, 982  
Quality of Service 70, 245  
Query Hot-Plug System Driver 875  
Query Slot Status 875  
quiesce 873  
Quiesce command 853  
Quiescing Card 873  
Quiescing Card and Driver 873  
Quiescing Driver 873

**R**

Rate ID 512  
Ratios 478  
Receive Buffer 403  
Receive Logic 366, 392  
Receiver Characteristics 492, 497  
Recovery Process 572  
Recovery State 520, 571  
Recovery State - Entry 572  
Recovery.Equalization 587  
Recovery.RcvrCfg 574, 575, 576, 598  
Recovery.RcvrLock 573, 576  
Recovery.Speed 575, 595  
Refclk 455  
Relaxed Ordering 286, 296, 299  
Replay Mechanism 74  
Replay Timer 690  
Request TLP 184  
Request Types 59  
Requester 33  
Requester ID 982  
Reset 846  
Resizable BARs 135, 911  
Resume command 853  
Retention Latch 861  
Retention Latch Sensor 861  
Retry 21  
RO 297  
Root Complex 91, 109, 147, 163, 668,

- 696, 812, 972, 982  
 Root Complex Error Status 696  
 Root Error Command Register 698  
 Routing Elements 147  
 Routing Mechanisms 155  
 RST# 854  
 RTL Simulation 918  
 Run Length 982  
 Rx Buffer 403  
 Rx Clock 435  
 Rx Clock Recovery 394, 437  
 Rx Equalization 493  
 Rx Preset Hint Encodings 580
- S**
- Scrambler 366, 377
  - Scrambler implementation 379
  - Scrambling 983
  - SDP 373, 387
  - Secondary Bus Reset 840
  - Sequence Number 326
  - Serial Transport 41
  - Serializer 389
  - Service Interval 279
  - Set Slot Power Limit Message 210
  - Set Slot Status 875
  - Severity of Error 693
  - Short Circuit Requirements 459
  - SHPC 1.0 848
  - SI 278
  - Signal Attenuation 485
  - Simplified Ordering Rule 287
  - Simplified Ordering Table 914
  - Single Host System 94
  - Single-Root System 113
  - SKIP 386, 387
  - SKIP ordered set 392
  - Skip Ordered Set 983
  - SKP 386
  - SKP Ordered Set 389
  - Slot Capabilities 865
  - Slot Capabilities Registers 865
  - Slot Control 868
  - Slot Control Register 869
  - Slot Numbering 862
  - Slot Numbering Identification 862
  - Slot Power Limit Control 867, 881
  - Slot Power Limit Message 210
  - Slot Status 870
  - Soft Off 708
  - SOS 389, 983
  - South Bridge 11
  - Spec Revision 2.1 887
  - Speed Change 568
  - Speed Change Example 576, 622
  - Speed Changes - Software 627
  - Split Transaction Protocol 149
- SR-IOV 937  
 SSC 983  
 SSC Modulation 455  
 SSD Modules 940  
 Start command 853  
 Sticky Bits 688, 983  
 STP 373, 387  
 Strict Priority VC Arbitration 253  
 Strong Ordering 286  
 Subordinate Bus Number register 93  
 Surprise Removal 849  
 Surprise Removal Notification 849  
 Switch 269, 278, 938, 971, 983  
 Switch Arbitration 269  
 Switch Port 57  
 Switch Routing 161  
 Switches 941  
 Symbol 366, 983  
 Symbol Lock 78, 396, 507, 983  
 Symbol time 983  
 Symbols 381  
 Sync Header 364  
 System PM States 708  
 System Reset 833
- T**
- Target 21, 22
  - TBWRR 266, 279
  - TC 247, 285, 287
  - TC to VC Mapping 249
  - TC/VC Mapping 248, 252
  - Time-Based, Weighted Round Robin Arbitration 266
  - TLP 60, 61, 170, 172, 984
  - TLP Elements 169
  - TLP Header 154
  - TLP Header Format 175
  - TLP Prefixes 908
  - TLP Processing Hints 899, 984
  - TLP Routing 145, 147
  - TLP Structure 174
  - Token 984
  - token 712
  - TPH 899, 900, 984
  - TPH Capability 907
  - TPH Control 907
  - Trace Viewer 924
  - Traffic Class 71, 174, 176, 183, 247, 248
  - Training Control 512
  - Training Examples 542
  - Training Sequence 1 369
  - Transaction Attributes 183
  - Transaction Descriptor 182
  - Transaction ID 183
  - Transaction Layer 55, 59
  - Transaction Layer Packet 60, 172
  - Transaction Ordering 71, 285

Transaction Routing 121  
Transaction Stalls 300  
Transactions 150  
Transactions Pending Buffer 228  
Translating Slot IDs 873  
Transmission Loss 468  
Transmit Logic 364, 368  
TS1 388  
TS1 and TS2 Ordered Sets 510  
TS1 Ordered-Set 842  
Turning Slot Off 855  
Turning Slot On 855  
Tx Buffer 368, 435  
Tx Clock 390  
Tx Equalization 448  
Tx Equalization Tolerance 448  
Tx Preset Encodings 579  
Tx Signal Skew 390  
Type 0 Configuration Request 99  
Type 1 Configuration Request 100  
Type 1 configuration transaction 93  
Type Field 179

## ***U***

UI 984  
Uncorrectable Error Reporting 694  
Uncorrectable Errors 984  
Uncorrectable Fatal Errors 652  
Uncorrectable Non-Fatal Errors 652  
Unexpected Completion 664  
Unit Interval 984  
Unlock Message 209  
Unsupported Request 663  
UpdateFC-Cpl 312  
UpdateFC-NP 312  
UpdateFC-P 312  
USB Bus Driver 711  
USP 984

## ***V***

Variables 985  
VC 216, 247, 287  
VC Arbitration 252, 257  
VC Buffers 301  
Vendor Specific 311  
Vendor Specific DLLP 311  
Vendor-Defined Message 210  
Virtual Channel 218, 258, 301  
Virtual Channel Arbitration Table 258  
Virtual Channel Capability Registers 246  
Virtual Channels 247

## ***W***

WAKE# Signal 772  
WAKE# signal 773  
Warm Reset 834

WDM Device Driver 706  
Weak Ordering 286, 299  
Weighted Round Robin Arbitration 256  
Weighted Round Robin Port Arbitration 265  
Weighted Round Robin VC Arbitration 257  
Working state 708  
Write Transaction 68  
WRR 256

# LeCroy

## World Leader in PCI Express® Protocol Test and Verification

LeCroy leads the protocol test and verification market with the most advanced and widest range of protocol test tools available on the market today. LeCroy's dedication to PCI Express development and test is demonstrated by our history of being first-to-market with new test capabilities to help you to be first-to-market with new PCI Express products. Among our accomplishments are:

- ✓ First PCIe® 1.0 Protocol Analyzer
- ✓ First PCIe 2.0 Protocol Analyzer
- ✓ First PCIe 2.0 Exerciser
- ✓ First PCIe 2.0 Protocol Test Card
- ✓ First PCIe 3.0 Protocol Analyzer
- ✓ First PCIe 3.0 Device Emulator

- ✓ First PCIe 3.0 Host Emulator
- ✓ First PCIe 3.0 Active Interposer
- ✓ First PCIe 3.0 MidBus Probe
- ✓ First PCIe 3.0 ExpressModule Interposer
- ✓ First to support NVM Express

LeCroy provides you the widest range of test tools and specialty probes to simplify and accelerate test and debug of all PCI Express products, providing tools with capabilities and price points to meet any customer's test requirements and budget.



Summit™ T3-16  
Protocol Analyzer



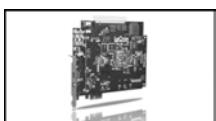
Summit T3-8  
Protocol Analyzer



Summit T2-16  
Protocol Analyzer



Summit T28  
Protocol Analyzer



Edge™ T1-4  
Protocol Analyzer



Summit Z3-16  
Device Emulator



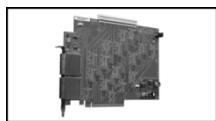
Summit Z3-16  
Host Emulator



Gen2 Protocol  
Test Card



SimPASS™ PE  
Simulation Analysis



Gen3 x16 Active  
Interposer



MidBus Probe



Multi-lead Probe



AMC Interposer



MiniCard Interposer

For many additional PCIe products and specialty probes, contact your local LeCroy representative or visit our website

For more information on LeCroy protocol verification solutions, please contact your Regional Sales Engineer: 1-800-909-7211 or 408-653-1262; or [PSGsales@lecroy.com](mailto:PSGsales@lecroy.com)

**[www.lecroy.com](http://www.lecroy.com)**

# MindShare Live Training and Self-Paced Training

<b>Intel Architecture</b> <ul style="list-style-type: none"><li>• Intel Ivy Bridge Processor</li><li>• Intel 64 (x86) Architecture</li><li>• Intel QuickPath Interconnect (QPI)</li><li>• Computer Architecture</li></ul>	<b>Virtualization Technology</b> <ul style="list-style-type: none"><li>• PC Virtualization</li><li>• IO Virtualization</li></ul>
<b>AMD Architecture</b> <ul style="list-style-type: none"><li>• AMD Opteron Processor (Bulldozer)</li><li>• AMD64 Architecture</li></ul>	<b>IO Buses</b> <ul style="list-style-type: none"><li>• PCI Express 3.0</li><li>• USB 3.0 / 2.0</li><li>• xHCI for USB</li></ul>
<b>Firmware Technology</b> <ul style="list-style-type: none"><li>• UEFI Architecture</li><li>• BIOS Essentials</li></ul>	<b>Storage Technology</b> <ul style="list-style-type: none"><li>• SAS Architecture</li><li>• Serial ATA Architecture</li><li>• NVMe Architecture</li></ul>
<b>ARM Architecture</b> <ul style="list-style-type: none"><li>• ARM Architecture</li></ul>	<b>Memory Technology</b> <ul style="list-style-type: none"><li>• Modern DRAM Architecture</li></ul>
<b>Graphics Architecture</b> <ul style="list-style-type: none"><li>• Graphics Hardware Architecture</li></ul>	<b>High Speed Design</b> <ul style="list-style-type: none"><li>• High Speed Design</li><li>• EMI/EMC</li></ul>
<b>Programming</b> <ul style="list-style-type: none"><li>• X86 Architecture Programming</li><li>• X86 Assembly Language Basics</li><li>• OpenCL Programming</li></ul>	<b>Surface-Mount Technology (SMT)</b> <ul style="list-style-type: none"><li>• SMT Manufacturing</li><li>• SMT Testing</li></ul>

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](mailto:training@mindshare.com)

**1-800-633-1440**

[www.mindshare.com](http://www.mindshare.com)



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

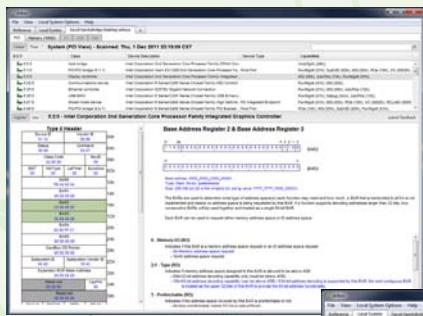
MindShare Arbor provides a very simple interface to directly edit a register in PCI config space, memory address space or IO address space. This can be done in the decoded view so you see what the meaning of each bit, or by simply writing a hex value to the target location.

## 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.



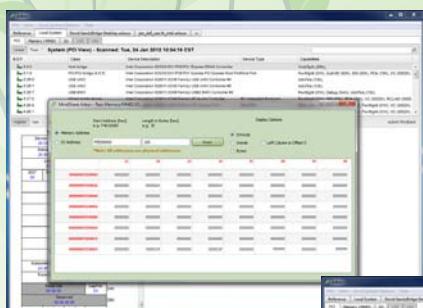
# The Ultimate Tool to View, Edit and Verify Configuration Settings of a Computer



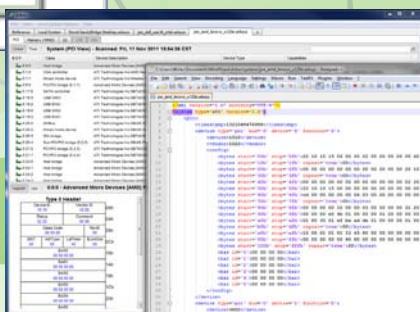
Decode Data from Live Systems



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.)