



# QuickPCle Expert for Xilinx

## Reference Manual

Version 1.5.6 February 2016  
Copyright © PLDA 1996-2016



# QuickPCIe Expert

## Reference Manual

### Documentation Change History

Date	Version Number	Changes
February 2016	1.5.6	<ul style="list-style-type: none"> <li>• Added Virtual Function Number signals to the AXI4-Lite Master and Slave interfaces.</li> </ul>
March 2015	1.5.5	<ul style="list-style-type: none"> <li>• Added Kintex-Ultrascale support to list of board features.</li> </ul>
December 2014	1.5.4	<ul style="list-style-type: none"> <li>• No change</li> </ul>
November 2014	1.5.3	<ul style="list-style-type: none"> <li>• Added user side band and data error signals for AXI Master and Slave interfaces.</li> <li>• Updated Device Configuration Core Constant signal names.</li> <li>• Added chapter on Protection Logic Interface</li> </ul>
March 2014	1.5.2	<ul style="list-style-type: none"> <li>• Added AXI4 Stream Out Reset Feature.</li> </ul>
November 2013	1.5.1	<ul style="list-style-type: none"> <li>• Added support for Artix-7 and Zynq 7000 FPGA Hard IP.</li> <li>• Added AXI4 Stream In Reset Feature.</li> </ul>
August 2013	1.5.0	<ul style="list-style-type: none"> <li>• Added support for Kintex-7 GTX FPGA Hard IP.</li> </ul>
June 2013	1.4.6	<ul style="list-style-type: none"> <li>• Added support for V7 Gen3 HIP.</li> </ul>
May 2013	1.4.5	<ul style="list-style-type: none"> <li>• Changed product name.</li> </ul>
December 2012	1.4.3	<ul style="list-style-type: none"> <li>• Added support for PCIe Identification Settings configurability.</li> <li>• Added support for Reference Clock Frequency configurability.</li> <li>• Added support for AXI Slave Fixed and Wrap Burst Types.</li> <li>• Added support for AXI Slave Narrow Transfers.</li> <li>• Added support for AXI Slave Non-Contiguous Strobes.</li> <li>• Added support for AXI Slave configurable ID Width.</li> <li>• Added support for Completion Ordering Rules.</li> </ul>
October 2012	1.4.2	<ul style="list-style-type: none"> <li>• Updated Bridge Configuration Space registers.</li> <li>• Updated DMA registers.</li> </ul>
August 2012	1.4.1	<ul style="list-style-type: none"> <li>• Added support for Stream In packing.</li> <li>• Added support for SG_DMA early termination with byte granularity.</li> <li>• Added support for Interconnect pipeline.</li> <li>• Revised document organization.</li> </ul>
June 2012	1.4.0	First release

### Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by PLDA SAS. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by PLDA in good faith. This document is provided "as is" with no warranties whatsoever, including any warranty of merchantability, non infringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample.

This document is intended only to assist the reader in the use of the product. PLDA shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product. Nor shall PLDA be liable for infringement of proprietary rights relating to use of information in this document. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

# Table of Contents

<b>List of Figures</b>	<b>9</b>
<b>List of Tables</b>	<b>10</b>
<b>Preface</b>	<b>12</b>
About this Document	12
Additional Reading	12
Feedback and Contact Information	12
<b>Chapter 1 QuickPCIe Expert Features</b>	<b>13</b>
<b>Chapter 2 QuickPCIe Expert Architecture</b>	<b>15</b>
2.1 PCI Express Layer	15
2.2 Bridge Layer	16
2.3 AXI Layer	16
<b>Chapter 3 Memory Specifications</b>	<b>17</b>
3.1 PCIe-QuickPCIe/QuickPCIe-PCIe Read Table Buffers	18
3.2 QuickPCIe DMA Buffers	19
3.3 QuickPCIe PCIe Address Translation Buffers	19
3.4 QuickPCIe AXI Address Translation Buffers	19
3.5 CDC Buffers	20
3.6 Memory Interfaces	21
<b>Chapter 4 Clocks and Resets</b>	<b>22</b>
<b>Chapter 5 Configuring the QuickPCIe Expert</b>	<b>24</b>
5.1 Configuring the QuickPCIe Expert Bridge IP	24
5.1.1 Opting for Hardwired or Reconfigurable Registers	24
5.1.2 Core Constant Parameters	25
5.1.2.1 Bridge General Settings Core Constants	25
5.1.2.2 Device Configuration Core Constants	25
5.1.2.3 PCI Express Configuration Core Constants	26
5.1.2.4 Bridge Interrupt Configuration Core Constants	26
5.1.2.5 Routing, Arbitration and Priority Rules Configuration Core Constants	27
5.1.2.6 AXI Interfaces Core Constants	29
5.1.2.7 DMA Engines Configuration Core Constants	34

	5.1.2.8	Address Translation Configuration Core Constants	37
5.2		Configuring the Bridge PCI Express Configuration Space	39
<b>Chapter 6</b>		<b>Xilinx 7 Series Hard IP Interface</b>	<b>40</b>
<b>Chapter 7</b>		<b>Programing a DMA Transfer</b>	<b>41</b>
7.1		Direct and Scatter-Gather DMA Basics	41
7.2		Transfer Parameters	42
7.3		Scatter-Gather Types	43
	7.3.1	SG_TYPE 00	43
	7.3.2	SG-TYPE 01	44
	7.3.3	SG_TYPE 10	44
	7.3.4	SG_TYPE 11	45
7.4		Scatter-Gather DMA Descriptors	46
7.5		Setting Start and End Conditions	48
7.6		Monitoring DMA	49
<b>Chapter 8</b>		<b>Implementing Address Translation</b>	<b>50</b>
8.1		Address Translation Mechanism	50
8.2		Setting Address Translation Tables	52
8.3		Monitoring Address Translation	52
<b>Chapter 9</b>		<b>AXI Interfaces</b>	<b>53</b>
9.1		AXI4-Lite Slave Interface	53
9.2		AXI4-Lite Master Interface	54
9.3		AXI4 Master Descriptor Interface	55
9.4		AXI4 Master Interfaces (0 - 3)	57
9.5		AXI4 Slave Interfaces (0 - 3)	59
9.6		AXI4 Stream Interfaces (0 - 3)	61
9.7		AXI4 Read and Write Burst Examples	62
<b>Chapter 10</b>		<b>Handling Interrupts</b>	<b>64</b>
10.1		Handling Host Interrupts	64
10.2		Handling Local Interrupts	64
10.3		Implementing MSI	65

<b>Chapter 11</b>	<b>Managing Low Power</b>	<b>66</b>
11.1	ASPM L0s Low Power	66
11.2	ASPM L1 Low Power	66
11.3	L1 Low Power	66
11.4	Handling Power Management Events	66
<b>Chapter 12</b>	<b>Test Interface</b>	<b>67</b>
<b>Chapter 13</b>	<b>Protection Logic Interface</b>	<b>67</b>
<b>Chapter 14</b>	<b>Bridge Configuration Space</b>	<b>68</b>
14.1	Control and Status Registers	69
14.2	Interrupt and Event Registers	92
14.3	Routing, Arbitration and Priority Rules Registers	96
14.4	Optional Features Registers	98
14.5	DMA Engines Registers	99
14.6	Address Translation Registers	104
14.7	PCI Express Configuration Space	105
14.8	Bridge External Registers	105
<b>Appendix A:</b>	<b>Register Content of the PCIe Config Space</b>	<b>106</b>
A.1	PCI Configuration Space	106
A.1.1	Type 0 Standard PCI Configuration Header	107
A.1.2	PCI Express Capability Structure	107
A.1.3	MSI Capability Structure	112
A.1.4	Power Management Capability Structure	114
A.2	PCI Express Extended Configuration Space	116
A.2.1	Vendor Specific Extended Capability Structure	117
A.2.2	Advanced Error Reporting Capability Structure	118
A.2.3	Secondary PCI Express Extended Capability Structure	120
<b>Appendix B:</b>	<b>Message and Event Handling</b>	<b>121</b>
B.1	Sending Messages to the PCI Express Interface	121
B.2	Received Message Processing	121
B.3	Event Processing	123

<b>Appendix C: AXI Interface Implementation</b>	<b>125</b>
C.1    AXI Implementation Notes	125
C.2    AXI Limitations	126
C.3    AXI4-Stream Limitations	127
 <b>Appendix D: PCI Express System Performance</b>	 <b>129</b>
D.1    Latency	129
D.2    Maximum Effective Bandwidth	130
D.3    Actual Link Usage	130
D.4    System Performance Illustrated	131
D.4.1    One Outstanding Request, Small Packet	131
D.4.2    Two Outstanding Requests, Small Packet	131
D.4.3    Four Outstanding Requests, Small Packet	132
D.4.4    Two Outstanding Requests, Large Packet	132
D.4.5    High Latency System	133
 <b>Appendix E: PCI Express Fundamentals</b>	 <b>134</b>
E.1    About PCI Express	134
E.2    PCI Express Lanes and Links	134
E.3    The PCI Express Fabric	136
E.3.1    Root Complex	136
E.3.2    Endpoint	137
E.3.3    Switch	137
E.3.4    Bridge	137
E.4    Types of Transactions	138
E.5    Routing Rules	139
E.5.1    Routing by Address	140
E.5.2    Routing by ID	140
E.5.3    Implicit Routing	140
E.5.4    Routing Examples	141
E.6    Flow Control	143
E.6.1    Traffic Classes (TCs)	143
E.6.2    Virtual Channels (VCs)	143
E.6.3    TCs and VCs	143
E.6.4    Receive Buffer	144
E.6.5    Flow Control Credits	144
E.6.6    Deadlock Avoidance	145
E.7    Error Handling	146
E.7.1    Error Recovery	146

E.7.2	End-to-End Cyclic Redundancy Check (ECRC) .....	146
E.7.3	Replay buffer.....	146
E.7.4	Completion Timeout.....	146



## List of Figures

Figure 1: QuickPCle Expert Architecture . . . . .	15
Figure 2: Bridge Interconnect connections . . . . .	16
Figure 3: Memory buffers . . . . .	17
Figure 4: Direct DMA Transfer . . . . .	41
Figure 5: SG-DMA Chained List . . . . .	42
Figure 6: SG_TYPE 00 . . . . .	43
Figure 7: SG_TYPE 01 . . . . .	44
Figure 8: SG_TYPE 10 . . . . .	44
Figure 9: SG_TYPE 11 . . . . .	45
Figure 10: PCIe to AXI4 Master Address Translation . . . . .	50
Figure 11: AXI4 Slave to PCIe Address Translation . . . . .	51
Figure 12: Read Burst Example 1 . . . . .	62
Figure 13: Read Burst Example 2 . . . . .	62
Figure 14: Write Burst Example . . . . .	63
Figure 15: One outstanding request, small packet. . . . .	131
Figure 16: Two outstanding requests, small packet. . . . .	131
Figure 17: Four outstanding requests, small packet . . . . .	132
Figure 18: Two outstanding requests, large packet . . . . .	132
Figure 19: High latency system . . . . .	133
Figure 20: A PCI Express Lane . . . . .	134
Figure 21: PCI Express x4 Link. . . . .	135
Figure 22: A Typical PCI Express Fabric . . . . .	136
Figure 23: Tracing a Write Transaction through the fabric . . . . .	141
Figure 24: Tracing a Read Transaction through the fabric . . . . .	142
Figure 25: Flow Control through Virtual Channels (VCs) and Traffic Classes (TCs). . . . .	143
Figure 26: Flow Control through a single Link . . . . .	143
Figure 27: Receive buffers for a Virtual Channel . . . . .	144

## List of Tables

Table 1: Memory Specifications . . . . .	17
Table 2: Bridge buffer interface signals . . . . .	21
Table 3: QuickPCIe Expert clock and reset signals . . . . .	22
Table 4: Core Constants - Bridge General Settings . . . . .	25
Table 5: Core Constants - Device Configuration . . . . .	25
Table 6: Core Constants - PCI Express Configuration . . . . .	26
Table 7: Core Constants - Bridge Interrupt Configuration . . . . .	26
Table 8: Core Constants - Routing, Arbitration and Priority Rules Configuration . . . . .	27
Table 9: Core Constants - AXI Interfaces . . . . .	29
Table 10: Core Constants - DMA Engines Configuration . . . . .	34
Table 11: Core Constants - Address Translation Configuration . . . . .	37
Table 12: Hard IP interface signals . . . . .	40
Table 13: Transfer Parameters . . . . .	42
Table 14: Scatter-Gather DMA Descriptors Mapping . . . . .	46
Table 15: AXI4-Lite Slave interface signals . . . . .	53
Table 16: AXI4-Lite Master interface signals . . . . .	54
Table 17: AXI4 Master Descriptor interface signals . . . . .	55
Table 18: AXI4 Master 0 interface signals . . . . .	57
Table 19: AXI4 Slave 0 interface signals . . . . .	59
Table 20: AXI4 Stream interface signals . . . . .	61
Table 21: Local interrupt signals . . . . .	65
Table 22: Test interface signals . . . . .	67
Table 23: Bridge Configuration Space . . . . .	68
Table 24: Control and Status Registers . . . . .	69
Table 25: Interrupt and Event Registers . . . . .	92
Table 26: Routing, Arbitration, and Priority Rules Registers . . . . .	96
Table 27: Optional Features Registers . . . . .	98
Table 28: DMA Engine Registers . . . . .	99
Table 29: DMA Engine Configuration Space Registers . . . . .	99
Table 30: Address Translation Registers . . . . .	104
Table 31: Address Translation Table Registers . . . . .	104
Table 32: PCI Configuration Space Layout . . . . .	106
Table 33: Type 0 Standard PCI Configuration Header . . . . .	107
Table 34: PCI Express Capability Structure . . . . .	107
Table 35: PCI Express Capability List Register . . . . .	108
Table 36: PCI Express Capabilities Register . . . . .	108
Table 37: Device Capabilities Register . . . . .	108
Table 38: Device Status Register . . . . .	109
Table 39: Link Capabilities Register . . . . .	109
Table 40: Link Status Register . . . . .	110
Table 41: Device Capabilities Register . . . . .	110
Table 42: Device Control 2 Register . . . . .	111
Table 43: Link Capabilities 2 Register . . . . .	111
Table 44: Message Signaled Interrupt (MSI) Capability Structure . . . . .	112
Table 45: Capability ID for MSI . . . . .	112
Table 46: Next Pointer for MSI . . . . .	112
Table 47: Message Control for MSI . . . . .	113
Table 48: Power Management Capability Structure . . . . .	114
Table 49: Capability ID . . . . .	114
Table 50: Next Item Pointer . . . . .	114
Table 51: Power Management Capabilities . . . . .	114
Table 52: PMCSR . . . . .	115
Table 53: Data Register . . . . .	115
Table 54: PCI Express Extended Configuration Space Layout . . . . .	116
Table 55: Vendor-Specific Extended Capability Structure . . . . .	117

<b>Table 56: Vendor-Specific Extended Capability Header</b> . . . . .	<b>117</b>
<b>Table 57: Vendor-Specific Header</b> . . . . .	<b>117</b>
<b>Table 58: Advanced Error Reporting Extended Capability Structure</b> . . . . .	<b>118</b>
<b>Table 59: AER Extended Capability Header</b> . . . . .	<b>118</b>
<b>Table 60: Advanced Error Capabilities and Control Register</b> . . . . .	<b>119</b>
<b>Table 61: Root Error Status Register</b> . . . . .	<b>119</b>
<b>Table 62: Secondary PCI Express Extended Capability Structure</b> . . . . .	<b>120</b>
<b>Table 63: Secondary PCI Express Extended Capability Header</b> . . . . .	<b>120</b>
<b>Table 64: Received Message Processing</b> . . . . .	<b>121</b>
<b>Table 65: Event Processing</b> . . . . .	<b>123</b>
<b>Table 66: AXI Limitations</b> . . . . .	<b>125</b>
<b>Table 67: AXI Limitations</b> . . . . .	<b>126</b>
<b>Table 68: AXI-Stream Limitations</b> . . . . .	<b>127</b>
<b>Table 69: Typical latency values of Read Request transactions</b> . . . . .	<b>129</b>
<b>Table 70: Sample Maximum Effective Bandwidth Values</b> . . . . .	<b>130</b>
<b>Table 71: Configuration of Switch ports</b> . . . . .	<b>137</b>
<b>Table 72: PCI Express transaction types and characteristics</b> . . . . .	<b>138</b>
<b>Table 73: Completion transaction characteristics</b> . . . . .	<b>139</b>
<b>Table 75: Configuration Write transaction steps</b> . . . . .	<b>141</b>
<b>Table 76: Memory Read transaction steps</b> . . . . .	<b>142</b>
<b>Table 77: Example of an Endpoint's advertised credits at and after Link initialization and the effect on Flow Control</b> . . . . .	<b>145</b>

## Preface

## About this Document

This document has been written for design managers, system engineers, and designers of FPGAs who are evaluating or using the PLDA QuickPCIe ExpertCore.

## Additional Reading

Please refer to the following documents for information on specification standards:

- AMBA® AXI Specification, Version 2.0 - ARM, March 2010
- AMBA® 4 AXI4 Stream Protocol Specification, Version 1.0 - ARM, March 2010
- PCI Express™ Base Specification Revision 3.0
- PCI Power Management Specification, v1.2

## Feedback and Contact Information

### Feedback about this Document

PLDA welcomes comments and suggestions pertaining to this documentation. Please contact PLDA Technical Support and provide the following information:

- the title of the document
- the page number to which your comments refer
- a description of your comments

### Contact information

#### Corporate Headquarters

PLDA  
805, avenue J.R.G.G. de la Lauzière  
13290 Aix-en-Provence  
France

Tel: USA +1 408 273 4528 - International +33 442 393 600  
Fax: +33 442 394 902

#### Sales

For sales questions, please contact [sales@plda.com](mailto:sales@plda.com).

#### Technical Support

For technical support questions, please contact PLDA Support at <https://www.plda.com/support> using the Support Center if you have a PLDA online account.

If you don't have a PLDA account, contact <https://www.plda.com/user/register>.

# Chapter 1 QuickPCle Expert Features

The QuickPCle Expert provides a high-performance, easy-to-use interconnect solution between PCI Express and the latest version of the AXI protocol. QuickPCle Expert inherits the leading architecture and reliability of PLDA's previous generations of PCI Express interface IP and features an AXI user interface with built-in DMA, compliant with the AMBA® AXI3 and AXI4 specifications.

The following table describes the features of the QuickPCle Expert:

PCle Interface	<ul style="list-style-type: none"> <li>• x8, x4, x2, or x1 at 8.0 Gbps link speed (Gen3) with Kintex-Ultrascale and Virtex-7 GTH FPGA Hard IP</li> <li>• x8, x4, x2, or x1 at 5.0 Gbps link speed (Gen2) with Kintex-Ultrascale, Virtex-7 GTH, Virtex-7 GTX, Kintex-7 GTX and Zynq-7045 FPGA Hard IP</li> <li>• x4, x2, or x1 at 5.0 Gbps link speed (Gen2) with Artix-7 and Zynq-7030 FPGA Hard IP</li> <li>• <i>PCI Express Base Specification Revision 3.0</i> compliant</li> <li>• Designed for Endpoint only</li> <li>• 256-Byte maximum payload size</li> <li>• 1 Virtual Channel (VC)</li> <li>• 1 Function</li> <li>• Advanced Error Reporting (AER) support</li> <li>• ECRC generation and check support</li> <li>• Lane reversal support</li> <li>• Legacy PCI Power Management support</li> <li>• Native Active State Power Management L0s and L1 state support</li> <li>• Power Management Event (PME message)</li> <li>• MSI (up to 32) and INT message support</li> <li>• Unused features not implemented</li> </ul>
AXI Interfaces	<ul style="list-style-type: none"> <li>• Optional AXI4-Lite Slave interface for Bridge Configuration</li> <li>• Optional AXI4-Lite Master interface for External Registers Configuration</li> <li>• Optional AXI4 Master Descriptor interface to access SG-DMA Descriptors in AXI domain</li> <li>• Up to 4 AXI4 Master interfaces, each supporting up to 16 outstanding read requests</li> <li>• Up to 4 AXI4 Slave interfaces, each supporting up to 16 outstanding read requests</li> <li>• Up to 4 AXI4 Stream Input and Output interfaces, each handling up to 8 TID/TDEST combinations simultaneously</li> <li>• 64-bit, 128-bit, or 256-bit data support for AXI4 Master, Slave, and Stream interfaces</li> <li>• By-passable CDC for AXI4 Master, Slave, and Stream interfaces</li> <li>• AXI4 Master and Slave interfaces can be configured as AXI3 interfaces</li> </ul>
Configuration	<ul style="list-style-type: none"> <li>• Bridge Configuration Space accessible by PCle and/or AXI4-Lite Slave Interface</li> <li>• 4 KBytes for Bridge Internal Registers</li> <li>• 4 KBytes for PCle Configuration Space</li> <li>• 8 KBytes dedicated to user-defined external registers on AXI domain</li> <li>• Bridge Internal Registers can be hardwired for a lower footprint</li> </ul>

## DMA Engines

- Up to 8 fully-independent DMA Engines
- Up to 4 GBytes or infinite length transfers
- Up to 16 outstanding read and write requests
- Completion re-ordering support
- Reconfigurable Source and Destination, enabling targeting of PCIe, AXI4 Master, or AXI Stream input and output interfaces
- Flexible Scatter-Gather DMA modes, including dynamic DMA control per Descriptor
- End of packet generation and early termination support when targeting Stream interfaces
- Optional DMA Engine reporting to the Descriptor to ease software management
- Fetching of up to 3 Descriptors to optimize throughput

## Address Translation

- Up to 16 reconfigurable address translation tables for PCIe interface
- Up to 8 reconfigurable address translation tables per AXI4 Slave interface
- Translated accesses can target PCIe, AXI4 Master or AXI4 Stream interfaces

## Chapter 2 QuickPCle Expert Architecture

The QuickPCle Expert consists of three different layers, as shown below; the PCI Express layer, the Bridge layer, and the AXI layer.

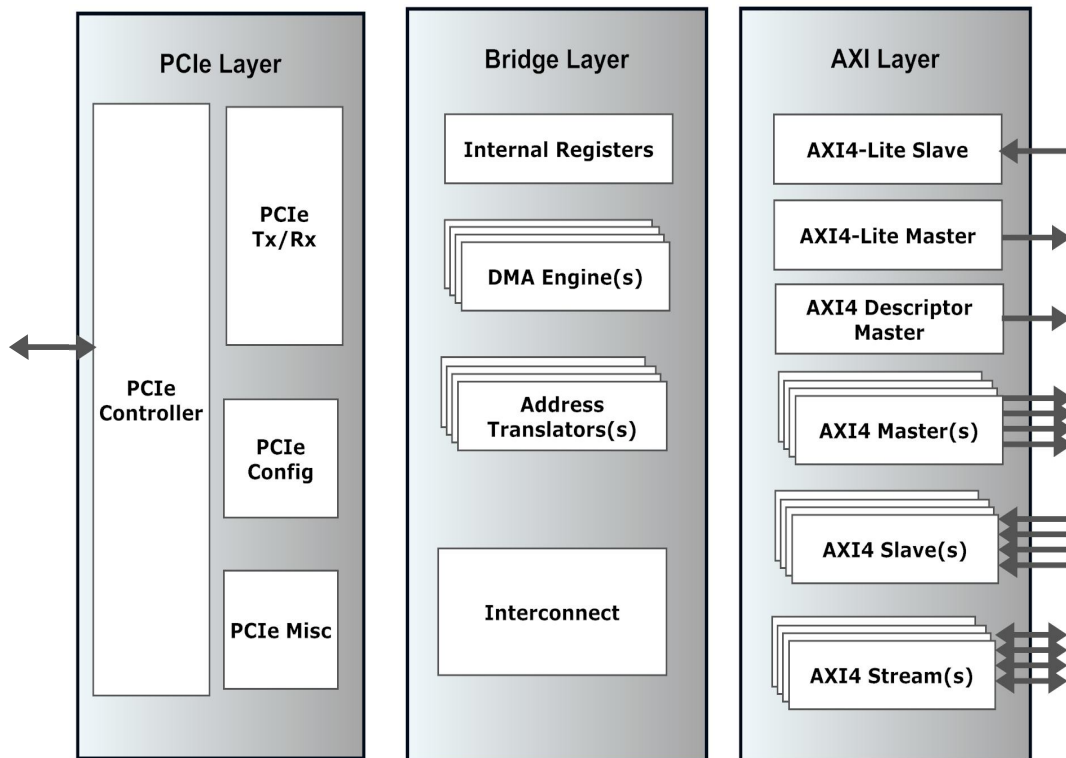


Figure 1: QuickPCle Expert Architecture

### 2.1 PCI Express Layer

The PCI Express layer consists of:

- a PCIe Controller, which is a Xilinx 7 Series Integrated Block for PCI Express
- a PCIe Tx/Rx Interface between the Bridge and the PCIe Controller
- a PCIe Configuration Interface to give the Bridge access to the PCIe Config Space
- a PCIe Misc Interface to allow the Bridge to manage Low-Power, Interrupts, etc.

The Core configures its memory space as follows:

- PCI Express BAR0/1 is configured as 64-bit prefetchable memory space of 16 KBytes. PCIe read and write requests targeting BAR0/1 are routed to the Bridge Configuration space (see [Chapter 14](#)).
- When PCIe Window #0 is implemented, PCI Express BAR2/3 is configured as a 64-bit prefetchable memory space of a size equal to PCIe Window #0. PCIe read and write requests targeting BAR2/3 are then routed to the PCIe Window #0 Address Translation Module.
- When PCIe Window #1 is implemented, PCI Express BAR4/5 is configured as a 64-bit prefetchable memory space of a size equal to PCIe Window #1. PCIe read and write requests targeting BAR4/5 are then routed to the PCIe Window #1 Address Translation Module.

## 2.2 Bridge Layer

The Bridge layer consists of:

- the Bridge Internal Registers (see [Chapter 14](#))
- up to 8 independent DMA Engine Modules (see [Chapter 7](#))
- Address Translator Modules to convert between the AXI and PCIe interfaces (see [Chapter 8](#))
- an Interconnect Module to interconnect and arbitrate between input and output flow

The following diagram shows how the Bridge Interconnect manages input and output flows:

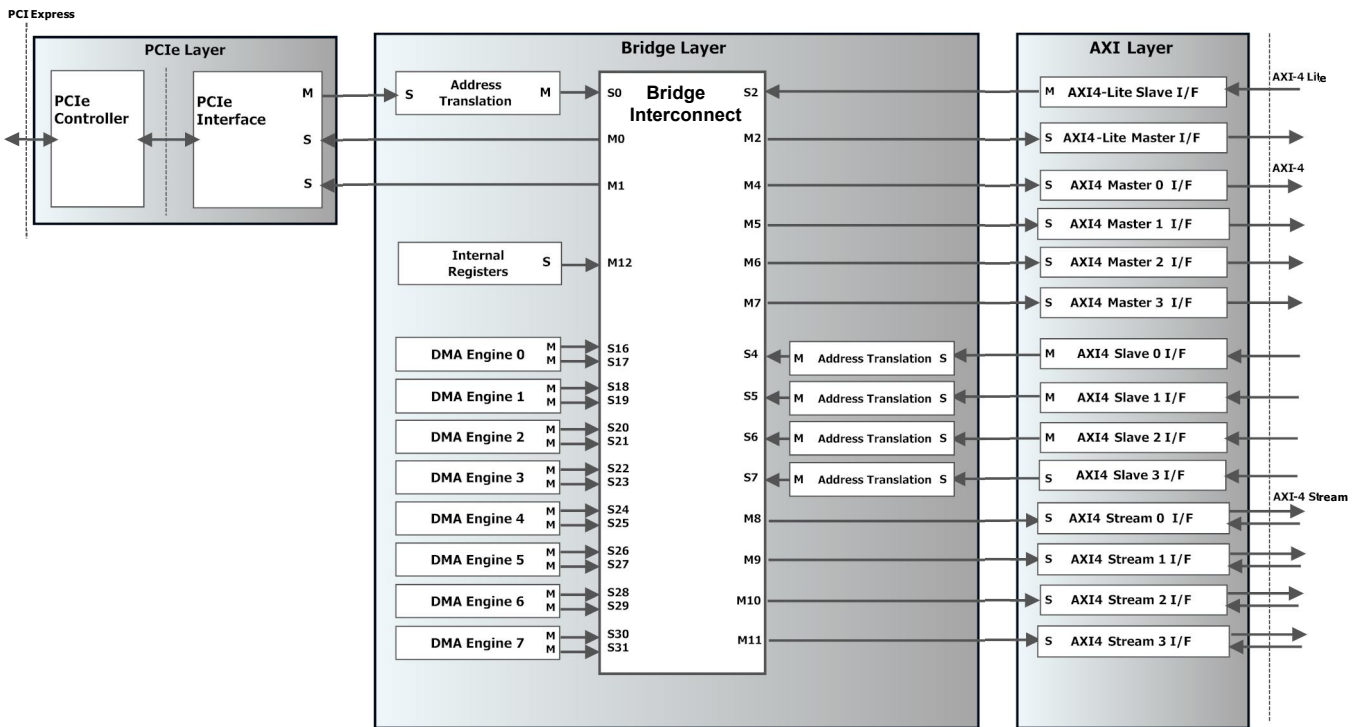


Figure 2: Bridge Interconnect connections

## 2.3 AXI Layer

The AXI layer consists of:

- an AXI4-Lite Slave Interface (for Bridge Configuration)
- an AXI4-Lite Master Interface (for External Registers Configuration)
- an AXI4 Master Descriptor Interface (to read SG-DMA Descriptors located in the AXI domain)
- up to 4 AXI4 Master Interfaces
- up to 4 AXI4 Slave Interfaces
- up to 4 AXI4 Stream In/Out Interfaces

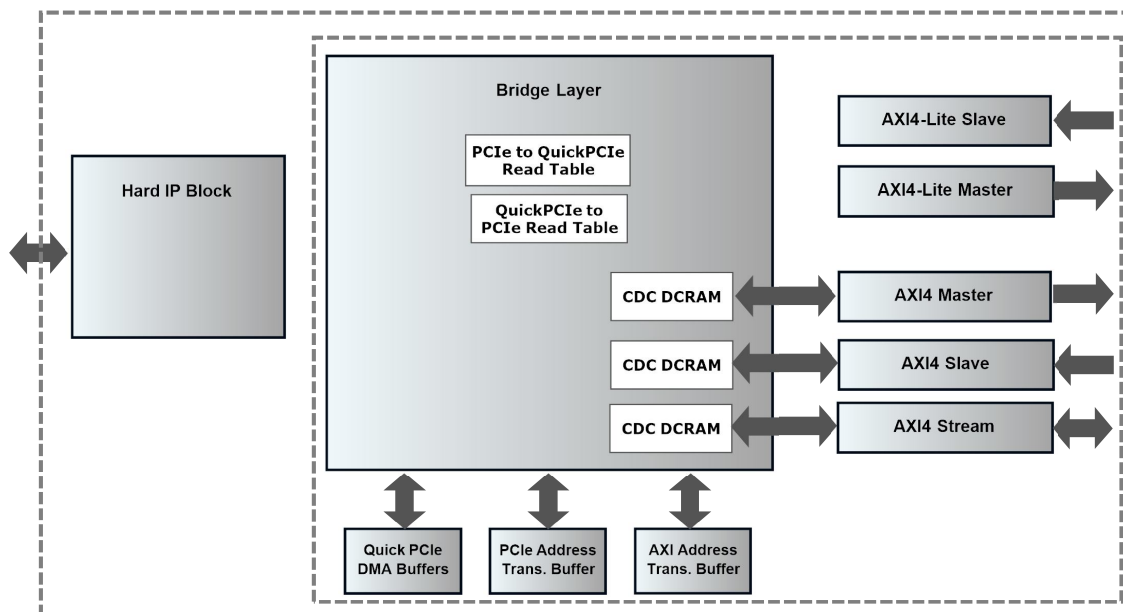
The AXI4 Master and Slave interfaces are assigned numbers 0 - 3, in the order in which they are implemented. For example, if only two AXI4 Master interfaces are implemented, these are Master interface 0 and Master interface 1. They cannot be labeled Master Interface 0 and Master Interface 3, for instance.

See [Chapter 9](#) for more information.



## Chapter 3 Memory Specifications

The following diagram illustrates the memories and buffers used by the QuickPCIe Expert Core:



**Figure 3: Memory buffers**

The following table describes each type of memory in the QuickPCIe Expert Core.

Memory	Bit Width	Depth (words)	No. of instances	RAM Type
PCIe to QuickPCIe Read Table	57	See <a href="#">Section 3.1</a>	1	Register file
QuickPCIe to PCIe Read Table	9	See <a href="#">Section 3.1</a>	1	Register file
QuickPCIe DMA buffers	256	See <a href="#">Section 3.2</a>	Equal to the number of DMA implemented	Single-clock dual-port RAM (1 write port and 1 read port)
PCIe Address Translation buffers	256	See <a href="#">Section 3.3</a>	2	Single-clock dual-port RAM (1 write port and 1 read port)
AXI Address Translation buffers	256	See <a href="#">Section 3.4</a>	Equal to 2 * number of Slave interfaces implemented	Single-clock dual-port RAM (1 write port and 1 read port)
CDC buffers	See <a href="#">Section</a>		1 per AXI4-Master, AXI4-Slave or AXI4-Stream Interface with CDC enabled	Register file(s)

**Table 1: Memory Specifications**

**Note:** Read latency for all memories is 1 clock cycle.

## 3.1 PCle-QuickPCle/QuickPCle-PCle Read Table Buffers

The depth of the PCle to QuickPCle Read Table and the QuickPCle to PCle Read Table buffers depends on whether the PCle Interface Window 0 or 1 is implemented (see the BRIDGE\_IMPL\_IF register in [Section 14.1](#)) or whether the buffers are used for DMA transfer only.

- If PCle Windows 0 or 1 are implemented:
  - PCle to QuickPCle Read Table = 12 words
  - QuickPCle to PCle Read Table = 9 words (for Virtex-6) or 12 words (for Virtex-7/Kintex-7)
- If PCle Windows 0 or 1 are not implemented:
  - PCle to QuickPCle Read Table = 4words
  - QuickPCle to PCle Read Table = 23 words (for Virtex-6/Virtex-7/Kintex-7)

## 3.2 QuickPCle DMA Buffers

The QuickPCle DMA buffers are used to fetch DMA transfers. One buffer is used for each implemented DMA Engine Module. Buffer size is equal to:

- $QPCIE\_DMAx\_OUTRREQ\_N \times QPCIE\_DMAx\_RREQ\_SIZE$

where  $QPCIE\_DMAx\_OUTRREQ\_N$  and  $QPCIE\_DMAx\_RREQ\_SIZE$  are respectively the Maximum Number of Outstanding Read Requests and the Maximum Read Request size that the DMA can issue.

## 3.3 QuickPCle PCIe Address Translation Buffers

The QuickPCle PCIe Address Translation buffers are used to fetch:

- PCIe to AXI write transactions
- AXI to PCIe read completions (in response to PCIe to AXI read transactions)

Their sizes are, respectively:

- $QPCIE\_ATR\_PCIE\_WBUF\_NUM \times QPCIE\_UNB\_MAXPAYLOAD$   
where  $QPCIE\_ATR\_PCIE\_WBUF\_NUM$  defines the number of buffers of Bridge Maximum Payload size ( $QPCIE\_UNB\_MAXPAYLOAD$ ) dedicated to storing PCI Express write requests
- $QPCIE\_ATR\_PCIE\_RBUF\_NUM \times QPCIE\_UNB\_MAXPAYLOAD$   
where  $QPCIE\_ATR\_PCIE\_RBUF\_NUM$  defines the number of buffers of Bridge Maximum Payload size ( $QPCIE\_UNB\_MAXPAYLOAD$ ) dedicated to storing AXI read completions for PCIe read requests.

## 3.4 QuickPCle AXI Address Translation Buffers

The QuickPCle AXI Address Translation buffers are used to fetch:

- AXI to PCIe write transactions
- PCIe to AXI read completions (in response to AXI to PCIe read transactions)

Their sizes are, respectively:

- $QPCIE\_ATR\_ASLV\_WBUF\_NUM \times QPCIE\_UNB\_MAXPAYLOAD$   
where  $QPCIE\_ASLV\_PCIE\_WBUF\_NUM$  defines the number of buffers of Bridge Maximum Payload size ( $QPCIE\_UNB\_MAXPAYLOAD$ ) dedicated to storing AXI write requests
- $QPCIE\_ATR\_ASLV\_RBUF\_NUM \times QPCIE\_UNB\_MAXPAYLOAD$   
where  $QPCIE\_ATR\_ASLV\_RBUF\_NUM$  defines the number of buffers of Bridge Maximum Payload size ( $QPCIE\_UNB\_MAXPAYLOAD$ ) dedicated to storing PCIe read completions for AXI read requests.

## 3.5 CDC Buffers

The size of each of the AXI interface CDC buffers, in registers, is equal to the AXI Interface width multiplied by the CDC FIFO Depth.

The AXI Interface width is equal to:

- 756 bits for a 256-bit data path AXI4 Master or AXI4 Slave interface
- 484 bits for a 128-bit data path AXI4 Master or AXI4 Slave interface
- 348 bits for a 64-bit data path AXI4 Master or AXI4 Slave interface
- 333 bits for a 256-bit data path AXI4 Stream In or AXI4 Stream Out interface
- 173 bits for a 128-bit data path AXI4 Stream In or AXI4 Stream Out interface
- 93 bits for a 64-bit data path AXI4 Stream In or AXI4 Stream Out interface

However, several registers are tied to 0 when certain AXI signals are not used.

For example:

- If addresses are aligned on 24-bits instead of 64-bits, the AXI Interface width is reduced by  $2 \times (64-24) = 80$  bits.
- If awqos/arqos signals are not used, AXI Interface width is reduced by  $2 \times 4 = 8$  bits.

The CDC FIFO Depth is equal to:

- 2^QPCIE\_AXI\_MSTx\_CDC\_CONF for AXI4 Master Interface
- 2^QPCIE\_AXI\_SLVx\_CDC\_CONF for AXI4 Slave Interface
- 2^QPCIE\_AXI\_STIx\_CDC\_CONF for AXI4 Stream In Interface
- 2^QPCIE\_AXI\_STOx\_CDC\_CONF for AXI4 Stream Out Interface

**Note:** A FIFO\_DEPTH of 2 will result in low performance, 4 will result in good performance if  $\text{Freq AXI} < \text{Freq Bridge}$ , and 16 will result in high performance.

## 3.6 Memory Interfaces

A memory interface is implemented for each RAM buffer, with the number of RAM buffers depending on the number of DMA Engines and Slave interfaces implemented in the Core.

In the following table, therefore, QPCIE\_RAM\_NUMBER = the sum of (QPCIE DMA Buffers + QPCIE PCIe Address Translation Buffers + QPCIE AXI Address Translation Buffers).

The interfaces for each buffer are the same.

Signal	I/O	Width	Description
<b>scram_wren</b>	out	[QPCIE_RAM_NUMBER-1:0]	Write Enable
<b>scram_wraddr</b>	out	[QPCIE_RAM_NUMBER*16-1:0]	Write Address
<b>scram_wrdata</b>	out	[QPCIE_RAM_NUMBER*QPCIE_DATAPATH-1:0]	Write Data
<b>scram_rden</b>	out	[QPCIE_RAM_NUMBER-1:0]	Read Enable
<b>scram_rdaddr</b>	out	[QPCIE_RAM_NUMBER*16-1:0]	Read Address
<b>scram_rddata</b>	in	[QPCIE_RAM_NUMBER*QPCIE_DATAPATH-1:0]	Read Data
<b>scram_rdderr</b>	in	[QPCIE_RAM_NUMBER*QPCIE_DATAPATH/64-1:0]	<p><b>Read Data Error:</b> This optional signal can be used to report that an error has been detected.</p> <p>It must be tied to 0s if the memory module has no data checking capability.</p> <p>Bit 0 reports errors on <code>SCRAM_RDDATA</code> bits 63:0.</p>

**Table 2: Bridge buffer interface signals**

## Chapter 4 Clocks and Resets

A Clock Domain Crossing (CDC) synchronizer can be implemented in each AXI interface module, enabling you to implement an independent clock domain for every AXI interface. Alternatively, you can use the same clock signal for each interface.

At power-up, both the QuickPCle Layer and the AXI Interfaces must be reset. The AXI resets can be deasserted at any time, while the QuickPCle Layer reset can be deasserted only when the `QPCIE_SRST_OUT` signal is deasserted.

If any strange behavior occurs in the AXI Interface, you can reset the AXI only if there are no read or write transactions pending, and if all DMA Engines attached to the AXI Interface are stopped. Otherwise a deadlock situation may occur.

The following table describes QuickPCle Expert clock and reset signals:

Signal	Description
<b>QuickPCle Layer signals</b>	
<code>qpcie_clk_out</code>	<b>QuickPCle Layer Clock:</b> This output signal is the clock for the QuickPCle Layer. The user application can use this clock when the CDC is bypassed.
<code>qpcie_clk_in</code>	<b>QuickPCle Layer Clock:</b> This input signal should be connected to the <code>QPCIE_CLK_OUT</code> signal.
<code>qpcie_srst_out</code>	<p><b>QuickPCle Layer Synchronous Reset Output:</b> This active-high synchronous reset signal indicates that the QuickPCle Layer logic should be reset. <code>QPCIE_RSTN</code> or <code>QPCIE_SRST_IN</code> must be kept asserted as long as <code>QPCIE_SRST_OUT</code> is asserted.</p> <p><b>Note:</b> You can use <code>QPCIE_SRST_OUT</code> to generate either <code>QPCIE_RSTN</code> or <code>QPCIE_SRST_IN</code> either by directly connecting the signals or by maintaining the Bridge Layer in reset for a longer time (for instance, if the application is waiting for a PLL lock).</p>
<code>qpcie_srst_in</code>	<b>QuickPCle Layer Synchronous Reset Input:</b> This active-high synchronous reset signal clears the QuickPCle Layer logic. It should be synchronous with the <code>QPCIE_CLK_IN</code> signal.
<code>qpcie_rstn</code>	<p><b>QuickPCle Layer Asynchronous Reset:</b> This active-low reset signal clears the QuickPCle Layer logic. Assertion of this signal can be asynchronous with <code>QPCIE_CLK_IN</code> but deassertion should be synchronous.</p> <p><b>Note:</b> You can use either <code>QPCIE_RSTN</code> or <code>QPCIE_SRST_IN</code> to perform a reset, but not both.</p>
<b>AXI4-Lite Slave signals</b>	
<code>axi4_slvl_aclk</code>	<b>AXI4-Lite Slave Clock:</b> The rising edge of the clock is the reference for all AXI4-Lite Slave signals.
<code>axi4_slvl_aresetn</code>	<b>AXI4-Lite Slave Asynchronous Reset:</b> This signal is the active-low reset signal for the Core associated with <code>AXI4_SLVL_ACLK</code> . Assertion of this signal can be asynchronous with <code>AXI4_SLVL_ACLK</code> but deassertion must be synchronous.
<code>axi4_slvl_asrst</code>	<b>AXI4-Lite Slave Synchronous Reset</b>
<b>AXI4-Lite Master signals</b>	
<code>axi4_mstl_aclk</code>	<b>AXI4-Lite Master Clock:</b> The rising edge of the clock is the reference for all AXI4-Lite Slave signals.
<code>axi4_mstl_aresetn</code>	<b>AXI4-Lite Master Asynchronous Reset:</b> This signal is the active-low reset signal for the Core associated with <code>AXI4_MSTL_CLK</code> . Assertion of this signal can be asynchronous with <code>AXI4_MSTL_CLK</code> but deassertion must be synchronous.
<code>axi4_mstl_asrst</code>	<b>AXI4 Master Synchronous Reset</b>
<b>AXI4 Master Descriptor signals</b>	
<code>axi4_mstd_aclk</code>	<b>AXI4 Master Descriptor Clock</b>
<code>axi4_mstd_aresetn</code>	<b>AXI4 Master Descriptor Asynchronous Reset</b>

Table 3: QuickPCle Expert clock and reset signals

Signal	Description
axi4_mstd_asrst	AXI4 Master Descriptor Synchronous Reset
AXI4 Master 0 Interface signals (also for Master Interfaces 1 - 3)	
axi4_mst0_aclk	AXI4 Master 0 Clock
axi4_mst0_aresetn	AXI4 Master 0 Asynchronous Reset
axi4_mst0_asrst	AXI4 Synchronous Reset
AXI4 Slave 0 Interface signals (also for Slave Interfaces 1 - 3)	
axi4_slv0_aclk	AXI4 Slave 0 Clock
axi4_slv0_aresetn	AXI4 Slave 0 Asynchronous Reset
axi4_slv0_asrst	AXI4 Synchronous Reset
AXI4 Stream Inputs and Outputs	
axi4_sto0_aclk	<b>AXI4 Stream 0 Output Clock:</b> The rising edge of the clock is the reference for all Stream Output signals. This signal is included in each of the Stream interfaces (0 - 3).
axi4_sti0_aclk	<b>AXI4 Stream 0 Input Clock:</b> The rising edge of the clock is the reference for all Stream Input signals. This signal is included in each of the Stream interfaces (0 - 3).
axi4_sto0_aresetn	AXI4 Stream 0 Output Asynchronous Reset
axi4_sti0_aresetn	AXI4 Stream 0 Input Asynchronous Reset
axi4_sto0_asrst	AXI4 Stream 0 Output Synchronous Reset
axi4_sti0_asrst	AXI4 Stream 0 Input Synchronous Reset

Table 3: QuickPCle Expert clock and reset signals

## Chapter 5 Configuring the QuickPCle Expert

### 5.1 Configuring the QuickPCle Expert Bridge IP

To configure the QuickPCle Expert Bridge IP, you must define Core Constants. The Core Constants are the basic Core settings defined when the Core is implemented. Unlike Core signals, constants cannot be dynamically modified.

We strongly recommend using the QuickPCle Expert Wizard to generate these constants. However, advanced users can generate specific QuickPCle Expert settings.

The Core Constants generated by the QuickPCle Expert Wizard are described in an include file called `qpcie-hip_coreconfig_h.v`.

They define:

- Bridge General Settings
- PCI Express Configuration
- Bridge Interrupt Fixed or Dynamic Configuration
- Routing, Arbitration and Priority Rules Static Configuration
- Routing, Arbitration and Priority Rules Fixed or Dynamic Configuration
- AXI Interfaces Static Configuration
- AXI Interfaces Fixed or Dynamic Configuration
- DMA Engines Static Configuration
- DMA Engines Fixed or Dynamic Configuration
- Address Translation Static Configuration
- Address Translation Fixed or Dynamic Configuration

#### 5.1.1 Opting for Hardwired or Reconfigurable Registers

QuickPCle Expert Core offers a programming mechanism that enables you to access and modify certain Core Constant settings.

These settings enable you to define:

- Arbitration and Priority Rules
- AXI Stream In Mode
- DMA Engines Options such as source, destination or SG support
- Address Translation Tables Settings

These Core Constants can be hardwired or reconfigured by registers. Hardwiring the settings allow gate count savings. When reconfigurable, the Core Constants can be modified by software.



## 5.1.2 Core Constant Parameters

The following tables describe each of the Core Constants in the QuickPCle Expert Core.

### 5.1.2.1 Bridge General Settings Core Constants

Constant	Supported Values	Description
QPCIE_DATAPATH	256 or 128	<b>Bridge Internal Data Path:</b> Either 256 bits when Core is in Gen3 mode, or 128 bits when Core is in Gen2 mode.
QPCIE_UNB_MAXPAYLOAD	128, 256, 512, 1024, 2048 or 4096	<b>Bridge Maximum Payload size:</b> The Maximum Payload size in bytes handled by the Bridge. If a packet of greater size is received on the PCI Express or AXI interface, it will be split by the Bridge into several packets with a maximum size equal to the Bridge's Maximum Payload size.
QPCIE_UNB_MAXRREQSIZE	128, 256, 512, 1024, 2048 or 4096	<b>Bridge Maximum Read Request size:</b> Maximum Read Request Size in bytes handled by the Bridge. If a read request of greater size is received on the PCI Express or AXI interface, the Bridge will split it into several read requests with a maximum size equal to the maximum size of the Bridge Read Request.

Table 4: Core Constants - Bridge General Settings

### 5.1.2.2 Device Configuration Core Constants

Constant	Supported Values	Description
<b>Device Configuration</b> The following Core Constants are Generics for VHDL and Parameters for Verilog.		
XXG_PCIE_VEN_ID	--	Vendor ID for the device
XXG_PCIE_DEV_ID	--	Device ID
XXG_PCIE_CLASS_CODE	--	Class code for the device
XXG_PCIE_SUB_VEN_ID	--	Sub Vendor ID for the device
XXG_PCIE_SUB_DEV_ID	--	Sub Device ID
XXG_PCIE__REFCLK_FREQ	--	Frequency of the Reference clock used by the Hard IP for the device.

Table 5: Core Constants - Device Configuration

### 5.1.2.3 PCI Express Configuration Core Constants

Constant	Supported Values	Description
<b>PCI Express Fixed or Dynamic Configuration</b>		
QPCIE_KPCIE_PM_CONF[95:0]	0 or 1	<b>Power Management Data Configuration Setting:</b> Defines the PCIe Core Power Management Data. See the PM_CONF register description in <a href="#">Section 14.1</a> .
QPCIE_IREG_RAZWI	1	<b>Bridge Internal Registers RAZ-WI:</b> When set to 1, the reserved Bridge Internal Registers are implemented as 'Read As Zero - Write Ignored'. This means that the registers are implemented as reading as all 0s, and writes to the registers are ignored.  Otherwise a write access to a read-only register or an unsupported value written to a writable register causes the Bridge to return a SLVERR condition on the AXI domain.

Table 6: Core Constants - PCI Express Configuration

### 5.1.2.4 Bridge Interrupt Configuration Core Constants

Constant	Supported Values	Description
<b>Bridge Interrupt Fixed or Dynamic Configuration</b>		
QPCIE_CFG_NO_SLVERR	0 or 1	<b>No Slave Error Response:</b> When set to 1, no SLVERR response is returned to the AXI or AXI4-Lite Slave interface when an Unsupported Request (UR) or a Configuration Request Retry Status (CRS) is returned in response to a PCIe Configuration Access.
QPCIE_KFIX_IMASK_LOCAL	0 or 1	<b>Local Interrupt Mask Programming:</b> When set to 1, the IMASK_LOCAL register (see <a href="#">Section 14.2</a> ) is hardwired. When set to 0, it can be dynamically reconfigured in the Bridge Configuration Space.
QPCIE_KINT_IMASK_LOCAL[31:0]	/	<b>Local Interrupt Mask Fixed Value:</b> When the QPCIE_KFIX_IMASK_LOCAL is set to 1, the IMASK_LOCAL register (see <a href="#">Section 14.2</a> ) is hardwired and equal to QPCIE_KINT_IMASK_LOCAL.
QPCIE_KFIX_IMASK_HOST	0 or 1	<b>Host Interrupt Mask Programming:</b> When set to 1, the IMASK_HOST register (see <a href="#">Section 14.2</a> ) is hardwired. When set to 0, it can be dynamically reconfigured in the Bridge Configuration Space.
QPCIE_KINT_IMASK_HOST[31:0]	/	<b>Host Interrupt Mask Fixed Value:</b> When QPCIE_KFIX_IMASK_HOST is set to 1, the IMASK_HOST register (see <a href="#">Section 14.2</a> ) is hardwired and equal to QPCIE_KINT_IMASK_HOST.

Table 7: Core Constants - Bridge Interrupt Configuration

## 5.1.2.5 Routing, Arbitration and Priority Rules Configuration Core Constants

Constant	Supported Values	Description
<b>Routing, Arbitration and Priority Rules Static Configuration</b>		
QPCIE_KARB00_RREQ_M[31:0] QPCIE_KARB15_RREQ_M[31:0]	/	<p><b>Read Request Routing Rules:</b> Defines from which requester a given device can receive a read request.</p> <p>When bit <i>j</i> of vector QPCIE_KARBi_RREQ_M[31:0] is asserted, it means that interface or module <i>i</i> is allowed to receive a read request from interface or module <i>j</i>, where the values of <i>i</i> and <i>j</i> can be:</p> <ul style="list-style-type: none"> <li>• 0: PCI Express interface</li> <li>• 1: PCI Express configuration space interface</li> <li>• 2: AXI4-Lite Slave or Master interface</li> <li>• 3: AXI4-Master Descriptor interface</li> <li>• 4,5,6,7: AXI4-Master or Slave #0,#1,#2,#3 interface</li> <li>• 8,9,10,11: AXI4-Stream In or Out #0,#1,#2,#3 interface</li> <li>• 12: Bridge Internal Registers</li> <li>• 13-15: reserved</li> <li>• 16,18,...,30: DMA Engine 0,1,...,DAM.fm7 Data interface</li> <li>• 17,19,...,31: DMA Engine 0,1,...,7 Descriptor interface</li> </ul> <p><b>Note:</b> DMA Engines can only issue read requests. They cannot receive read requests. Therefore, QPCIE_KARB16-31_RREQ_M is hardwired to 32'b0.</p>
QPCIE_KARB00_WREQ_M[31:0] QPCIE_KARB15_WREQ_M[31:0]	/	<p><b>Write Request Routing Rules:</b> Defines from which requester a given device can receive a write request.</p> <p>When bit <i>j</i> of vector QPCIE_KARBi_WREQ_M[31:0] is asserted, it means that interface or module <i>i</i> is allowed to receive a write request from interface or module <i>j</i>.</p> <p><b>Note:</b> DMA Engines can only issue write requests. They cannot receive write requests. Therefore, QPCIE_KARB16-31_WREQ_M is hardwired to 32'b0.</p>
QPCIE_PIPE_INTERCO_AUTO	undefined or 1 - 31	<p><b>Interconnect Automatic Pipeline:</b></p> <p>When defined, the Interconnect module is automatically pipelined. When arbitration is implemented between a number of requests greater than or equal to the value of QPCIE_PIPE_INTERCO_AUTO, the arbitrated request is pipelined.</p>
QPCIE_PIPE_INTERCONNECT	undefined or 1	<p><b>Interconnect Module Pipeline:</b></p> <p>When set to 1 and when QPCIE_PIPE_INTERCO_AUTO is undefined, all the outputs of the Interconnect module are pipelined.</p>
QPCIE_PIPE_INTERCO_RREQ	undefined or 1	<p><b>Interconnect Custom Pipeline Enable for Read Request:</b></p> <p>When QPCIE_PIPE_INTERCONNECT and QPCIE_PIPE_INTERCO_RREQ are set to 1 and QPCIE_PIPE_INTERCO_AUTO is undefined, custom pipelines are applied to Read Requests.</p>

Table 8: Core Constants - Routing, Arbitration and Priority Rules Configuration

Constant	Supported Values	Description
QPCIE_PIPE_UNB_RREQ[127:0]	/	<b>Interconnect Custom Pipeline for Read Requests:</b> Defines the Custom Pipeline for Read Requests: for Module i from 0 to 31: <ul style="list-style-type: none"> <li>• When bit <math>4*i + 3</math> is 1b, Request from the Interconnect to Module i is pipelined</li> <li>• When bit <math>4*i + 2</math> is 1b, Acknowledge from Module to Interconnect is pipelined</li> <li>• When bit <math>4*i + 1</math> is 1b, Request from Module to Interconnect is pipelined</li> <li>• When bit <math>4*i + 0</math> is 1b, Acknowledge from Interconnect to Module is pipelined</li> </ul>
QPCIE_PIPE_INTERCO_RCPL	undefined or 1	<b>Interconnect Custom Pipeline Enable for Read Completions:</b> When QPCIE_PIPE_INTERCONNECT and QPCIE_PIPE_INTERCO_RCPL are set to 1 and QPCIE_PIPE_INTERCO_AUTO is undefined, custom pipelines are applied to Read Completions.
QPCIE_PIPE_UNB_RCPL[127:0]	/	<b>Interconnect Custom Pipeline for Read Completions:</b> Defines the Custom Pipeline for Read Completions: for Module i from 0 to 31: <ul style="list-style-type: none"> <li>• When bit <math>4*i + 3</math> is 1b, Completion from Interconnect to Module i is pipelined</li> <li>• When bit <math>4*i + 2</math> is 1b, Acknowledge from Module to Interconnect is pipelined</li> <li>• When bit <math>4*i + 1</math> is 1b, Completion from Module to Interconnect is pipelined</li> <li>• When bit <math>4*i + 0</math> is 1b, Acknowledge from Interconnect to Module is pipelined</li> </ul>
QPCIE_PIPE_INTERCO_WREQ	undefined or 1	<b>Interconnect Custom Pipeline Enable for Write Requests:</b> When QPCIE_PIPE_INTERCONNECT and QPCIE_PIPE_INTERCO_WREQ are set to 1 and QPCIE_PIPE_INTERCO_AUTO is undefined, custom pipelines are applied to Write Requests.
QPCIE_PIPE_UNB_WREQ[127:0]	/	<b>Interconnect Custom Pipeline for Write Requests:</b> Defines the Custom Pipeline for Write Requests: for Module i from 0 to 31: <ul style="list-style-type: none"> <li>• When bit <math>4*i + 3</math> is 1b, Request from Interconnect to Module i is pipelined</li> <li>• When bit <math>4*i + 2</math> is 1b, Acknowledge from Module to Interconnect is pipelined</li> <li>• When bit <math>4*i + 1</math> is 1b, Request from Module to Interconnect is pipelined</li> <li>• When bit <math>4*i + 0</math> is 1b, Acknowledge from Interconnect to Module is pipelined.</li> </ul>
QPCIE_PIPE_INTERCO_WCPL	undefined or 1	<b>Interconnect Custom Pipeline Enable for Write Completions:</b> When QPCIE_PIPE_INTERCONNECT and QPCIE_PIPE_INTERCO_WCPL are set to 1 and QPCIE_PIPE_INTERCO_AUTO is undefined, custom pipelines are applied to Write Completions.

Table 8: Core Constants - Routing, Arbitration and Priority Rules Configuration

Constant	Supported Values	Description
QPCIE_PIPE_UNB_WCPL[127:0]	/	<b>Interconnect Custom Pipeline for Write Completions:</b> Defines the Custom Pipeline for Write Completions: for Module i from 0 to 31: <ul style="list-style-type: none"> <li>• When bit <math>4*i + 3</math> is 1b, Completion from Interconnect to Module i is pipelined</li> <li>• When bit <math>4*i + 2</math> is 1b, Acknowledge from Module to Interconnect is pipelined</li> <li>• When bit <math>4*i + 1</math> is 1b, Completion from Module to Interconnect is pipelined</li> <li>• When bit <math>4*i + 0</math> is 1b, Acknowledge from Interconnect to Module is pipelined</li> </ul>
<b>Routing, Arbitration and Priority Rules Fixed or Dynamic Configuration</b>		
QPCIE_KFIX_ARB_ARBTTYPE	0 or 1	<b>Arbitration Rules Programming:</b> When set to 1, the ARBITRATION_RULES registers (see <a href="#">Section 14.3</a> ) are hardwired. When set to 0, they can be dynamically reconfigured in the Bridge Configuration Space.
QPCIE_KARB_ARBTTYPE[511:0]	{{(32){16'h1111}}	<b>Arbitration Rules Fixed Value:</b> When QPCIE_KFIX_ARB_ARBTTYPE is set to 1, the ARBITRATION_RULES registers (see <a href="#">Section 14.3</a> ) are hardwired and equal to QPCIE_KARB_ARBTTYPE.
QPCIE_KFIX_ARB_PRIORITY	0 or 1	<b>Priority Rules Programming:</b> When set to 1, the PRIORITY_RULES registers (see <a href="#">Section 14.3</a> ) are hardwired. When set to 0, they can be dynamically reconfigured in the Bridge Configuration Space.
QPCIE_KARB_PRIORITY	{{(32){16'h0000}}	<b>Priority Rules Fixed Value:</b> When QPCIE_KFIX_ARB_PRIORITY is 1, the PRIORITY_RULES registers (see <a href="#">Section 14.3</a> ) are hardwired and equal to QPCIE_KARB_PRIORITY.

Table 8: Core Constants - Routing, Arbitration and Priority Rules Configuration

### 5.1.2.6 AXI Interfaces Core Constants

Constant	Supported Values	Description
<b>AXI Interfaces Static Configuration</b>		
QPCIE_AXI_SLVL_PRESENT	0 or 1	<b>AXI4-Lite Slave Interface Implementation:</b> When set to 1, the AXI4-Lite Slave interface is implemented.
QPCIE_AXI_MSTL_PRESENT	0 or 1	<b>AXI4-Lite Master Interface Implementation:</b> When set to 1, the AXI4-Lite Master interface is implemented.
QPCIE_AXI_MSTD_PRESENT	0 or 1	<b>AXI4 Master Descriptor Interface Implementation:</b> When set to 1, the AXI4 Master Descriptor interface is implemented.
QPCIE_AXI_MST_NUM	0, 1, 2, 3, 4	<b>AXI4 Master Interface Number:</b> Defines the number of AXI4 Master interfaces.
QPCIE_AXI_SLV_NUM	0, 1, 2, 3, 4	<b>AXI4 Slave Interface Number:</b> Defines the number of AXI4 Slave interfaces.

Table 9: Core Constants - AXI Interfaces

Constant	Supported Values	Description
QPCIE_AXI_STR_NUM	0, 1, 2, 3, 4	<b>AXI4 Stream Interface Number:</b> Defines the number of AXI4 Stream In or Out interfaces. It is equal to the number of Stream Out interfaces if the number of Stream Out interfaces is greater than the number of Stream In interfaces. If not, it is equal to the number of Stream In interfaces.
QPCIE_AXI_STRO_NUM	0, 1, 2, 3, 4	<b>AXI4 Stream Out Interface Number:</b> Defines the number of AXI4 Stream Out interfaces.
QPCIE_AXI_STRI_NUM	0, 1, 2, 3, 4	<b>AXI4 Stream In Interface Number:</b> Defines the number of AXI4 Stream In interfaces.
QPCIE_AXI_NARROW_ENABLE	0 or 1	<b>Narrow Transfer Enable:</b> When set to 1, AXI Slave Narrow Transfers are supported. See the <i>AMBA AXI Protocol Specification</i> for a description of narrow transfers.
QPCIE_AXI_MSTL_DATAPATH	32	<b>AXI4-Lite Master Data Path:</b> Fixed to 32 bits.
QPCIE_AXI_SLVL_DATAPATH	32	<b>AXI4-Lite Slave Data Path:</b> Fixed to 32 bits.
QPCIE_AXI_MSTD_TYPE QPCIE_AXI_MST0-3_TYPE QPCIE_AXI_SLV0-3_TYPE	0 or 1	<b>AXI4 Interface Type:</b> When set to 1, the interface is compliant with the AXI4 specification. When 0, it is compliant with the AXI3 specification.
QPCIE_AXI_MSTD_CDC_CONF QPCIE_AXI_MST0-3_CDC_CONF QPCIE_AXI_SLV0-3_CDC_CONF	0 - 4	<b>AXI4 CDC Implementation:</b> This constant is used to implement the CDC in the interface. When set to 0, the CDC is bypassed. When set to 1, the CDC is implemented and its FIFO depth is equal to $2^{\text{QPCIE\_AXI\_xxxx\_CDC\_CONF}}$ . We recommend a value of 4 to guarantee the best throughput.
QPCIE_AXI_MSTD_DATAPATH QPCIE_AXI_MST0-3_DATAPATH QPCIE_AXI_SLV0-3_DATAPATH	64, 128 or 256	<b>AXI4 Data Path</b>
QPCIE_AXI_MSTD_R_OUTREQ QPCIE_AXI_MST0-3_R_OUTREQ	2 - 16	<b>AXI4 Master Outstanding Read Requests Number:</b> Maximum number of Outstanding Read Requests that the Bridge can issue to the AXI4 Master interface.
QPCIE_AXI_MSTD_W_OUTREQ QPCIE_AXI_MST0-3_W_OUTREQ	2 - 16	<b>AXI4 Master Outstanding Write Requests Number:</b> Maximum number of Outstanding Write Requests that the Bridge can issue to the AXI4 Master interface.

Table 9: Core Constants - AXI Interfaces

Constant	Supported Values	Description
QPCIE_AXI_MSTD_MAX_PAY QPCIE_AXI_MST0-3_MAX_PAY	0 - 5	<p><b>AXI4 Master Maximum Payload size:</b> Maximum Payload size in bytes that the Bridge can issue to the AXI4 Master interface. Supported values are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> <li>• 14: 32 bytes</li> <li>• 15: 64 bytes</li> </ul> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>• The Bridge cannot issue packets greater than the Bridge Maximum Payload size (QPCIE_UNB_MAXPAYLOAD).</li> <li>• When the AXI4 interface type is AXI3, then the AXI4 Master Maximum Payload size must not exceed 16 x (AXI4 Data Path in bytes).</li> <li>• When AXI4 interface type is AXI4, then the AXI4 Master Maximum Payload size should not exceed 256 x (AXI4 Data Path in bytes).</li> <li>• This means that: <math>QPCIE\_AXI\_MSTx\_MAX\_PAY = \min(QPCIE\_UNB\_MAXPAYLOAD; 16 \text{ or } 256 \times QPCIE\_AXI\_MSTx\_DATAPATH/8)</math>.</li> </ul>
QPCIE_AXI_MSTD_MAX_RDS, QPCIE_AXI_MST0-3_MAX_RDS	0 - 5	<p><b>AXI4 Master Maximum Read Request Size:</b> Maximum Read Request size in bytes that the Bridge can issue to the AXI4 Master interface. Supported values are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> <li>• 14: 32 bytes</li> <li>• 15: 64 bytes</li> </ul> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>• The Bridge cannot issue read requests greater than the maximum size of the Bridge Read Request (QPCIE_UNB_MAXRREQSIZE).</li> <li>• When the AXI4 interface type is AXI3, then the maximum size of the AXI4 Master Read Request should not exceed 16 x (AXI4 Data Path in bytes).</li> <li>• When the AXI4 Interface Type is AXI3, then the maximum size of the AXI4 Master Read Request should not exceed 256 x (AXI4 Data Path in bytes).</li> <li>• This means that: <math>QPCIE\_AXI\_MSTx\_MAX\_PAY = \min(QPCIE\_UNB\_MAXPAYLOAD; 16 \text{ or } 256 \times QPCIE\_AXI\_MSTx\_DATAPATH/8)</math>.</li> </ul>
QPCIE_AXI_SLV0-3_R_OUTREQ	2 - 16	<p><b>AXI4 Slave Maximum Outstanding Read Requests:</b> Maximum number of Outstanding Read Requests from the AXI4 Slave interface that the Bridge can handle. Other Read Requests will be throttled.</p>

Table 9: Core Constants - AXI Interfaces

Constant	Supported Values	Description
QPCIE_AXI_SLV0-3_W_OUTREQ	2 - 16	<b>AXI4 Slave Maximum Outstanding Write Requests:</b> Maximum number of Outstanding Write Requests from the AXI4 Slave Interface that the Bridge can handle. Other Write Requests will be throttled.
QPCIE_AXI_SLV0-3_ID_WIDTH	1 - 24	<b>AXI Slave Transaction ID Width:</b> Specifies the AXI Slave Transaction ID fields in bits.
QPCIE_AXI_SLV0-3_MAX_PAY	0 - 5	<p><b>AXI4 Slave Maximum Payload size:</b> Maximum Payload size in bytes that the Bridge can receive from the AXI4 Slave interface. Supported values are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>• When the AXI4 interface type is AXI3, then the AXI4 Slave Maximum Payload size cannot exceed 16 x (AXI4 Data Path in bytes).</li> <li>• When the AXI4 interface type is AXI4, then the AXI4 Master Maximum Payload size cannot exceed 256 x (AXI4 Data Path in bytes).</li> <li>• This means that: QPCIE_AXI_SLVx_MAX_PAY = MIN(4096; 16 or 256 x QPCIE_AXI_SLVx_DATAPATH/8).</li> </ul>
QPCIE_AXI_SLV0-3_MAX_RDS	0 - 5	<p><b>AXI4 Slave Maximum Read Request size:</b> Maximum Read Request size in bytes that the Bridge can receive from the AXI4 Slave interface. Supported values are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>• When the AXI4 interface type is AXI3, then the AXI4 Master Maximum Read Request Size cannot exceed 16 x (AXI4 Data Path in bytes).</li> <li>• When the AXI4 Interface Type is AXI3, then the AXI4 Master Read Request size cannot exceed 256 x (AXI4 Data Path in bytes).</li> <li>• This means that: QPCIE_AXI_SLVx_MAX_PAY = MIN(4096; 16 or 256 x QPCIE_AXI_SLVx_DATAPATH/8).</li> </ul>
QPCIE_AXI_STO0-3_CDC_CONF QPCIE_AXI_STI0-3_CDC_CONF	0 - 4	<b>AXI4 Stream CDC Implementation:</b> This constant is used to implement the CDC in the interface. If set to 0, the CDC is bypassed. Otherwise, the CDC is implemented and its FIFO depth is equal to $2^{\text{QPCIE\_AXI\_STxx\_CDC\_CONF}}$ . We recommend a value of 4 to guarantee effective throughput.
QPCIE_AXI_STO0-3_DATAPATH QPCIE_AXI_STI0-3_DATAPATH	64, 128 or 256	<b>AXI4 Stream Data Path</b>

Table 9: Core Constants - AXI Interfaces



Constant	Supported Values	Description
QPCIE_AXI_STR0-3_R_OUTREQ	2 - 16	<p><b>AXI4 Stream In Outstanding Read Requests Number:</b> Maximum number of Outstanding Read Requests from the Bridge that the Stream In Interface can handle per TID/TDEST resource number.</p> <p>An AXI4 Stream In interface will accept data from the AXI4 Stream In bus only if this data is requested by an interface or a module (for example, if it is requested by a DMA Engine).</p>
QPCIE_AXI_STR0-3_IDDEST_N	1 - 8	<p><b>AXI4 Stream In TID/TDEST Resource Number:</b> An AXI4 Stream In interface can accept up to 8 separate data flows identified by different TID/TDEST.</p> <p>QPCIE_AXI_STRx_IDDEST_N should be set equal to the number of separate data flows the AXI4 Stream In is expected to receive.</p> <p><b>Note:</b> if QPCIE_AXI_STR0_IDDEST_N is greater than 1, the QPCIE_AXI_STRx_R_OUTREQ parameter must at least equal the maximum number of Outstanding Read Requests for the interface or module that is requesting data from the AXI4 Stream In interface, in order to prevent deadlock issues.</p>
QPCIE_AXI_STR0-3_A_LENGTH	0, 1, 2 or 3	<p><b>AXI4 Stream In Supported Length:</b> This constant defines the length of the packet that the AXI4 Stream In bus expects to receive. Supported values are:</p> <ul style="list-style-type: none"> <li>• 0: all stream lengths are supported</li> <li>• 1: length is a multiple of 4 bytes</li> <li>• 2: length is a multiple of 8 bytes</li> <li>• 3: length is a multiple of Bridge Data Path (QPCIE_DATAPATH)</li> </ul> <p>Restricting the address alignment reduces the Core gate count and allows higher frequency to be achieved.</p> <p>However, it also has an impact on the byte qualifiers supported by the AXI4 Stream In interface (see <a href="#">Appendix C.2</a>).</p>
QPCIE_AXI_STR0-3_A_RREQ	0, 1, 2 or 3	<p><b>AXI4 Stream In Supported Read Request Size:</b> This constant defines the size of the read request that the AXI4 Stream expects to receive. Supported values are:</p> <ul style="list-style-type: none"> <li>• 0: all read request sizes are supported</li> <li>• 1: read request size is a multiple of 4 bytes</li> <li>• 2: read request size is a multiple of 8 bytes</li> <li>• 3: read request size is a multiple of Bridge Data Path (QPCIE_DATAPATH)</li> </ul> <p>Restricting the address alignment reduces the Core gate count and enables higher frequency.</p>
QPCIE_AXI_STI3_PACKING	0 or 1	<p><b>AXI4 Stream In Packing Support:</b></p> <p>When set to 1, Stream In Packing is supported; non-contiguous strobes are supported. Null bytes and position bytes are removed from the data flow.</p>
<b>AXI Interfaces Fixed or Dynamic Configuration</b>		
QPCIE_KFIX_AXISTR0-3_CONF	1'b0 or 1'b1	<p><b>AXI4 Stream In Optional Settings Programming:</b> When set to 1, AXI4_STRINx_MODE registers (see <a href="#">Section 14.4</a>) are hardwired. When set to 0, they can be dynamically reconfigured in the Bridge Configuration Space.</p>

Table 9: Core Constants - AXI Interfaces

Constant	Supported Values	Description
QPCIE_AXI_STR0-3_T0-7_CONF[15:0]	/	<b>AXI4 Stream In Optional Settings Fixed Value:</b> When QPCIE_KFIX_AXISTR <sub>x</sub> _CONF is 1, AXI4_STRIN <sub>x</sub> _MODE registers (see <a href="#">Section 14.4</a> ) are hardwired and equal to QPCIE_AXI_STR <sub>x</sub> _T0-7_CONF[15:0].

Table 9: Core Constants - AXI Interfaces

### 5.1.2.7 DMA Engines Configuration Core Constants

Constant	Supported Values	Description
<b>DMA Engines Static Configuration</b>		
QPCIE_DMA_ENGINE_NUMBER	0 - 8	<b>DMA Engine Number:</b> This constant is used to specify the DMA Engine number. If set to 0, no DMA Engines are implemented.
QPCIE_DMA0-7_OUTRREQ_N[3:0]	4'd1 - 4'd15	<b>DMA Outstanding Read Request Number:</b> Defines the number of outstanding read requests the DMA Engine can issue to the source interface of the DMA transfer. Supported values are: <ul style="list-style-type: none"> <li>• 4'd1: Two outstanding read requests</li> <li>• ...</li> <li>• 4'd15: 16 outstanding read requests</li> </ul> The higher this value, the better performance will be, but buffer size will also be bigger.
QPCIE_DMA0-7_RREQ_SIZE[3:0]	4'd0 - 4'd15	<b>DMA Maximum Read Request Size:</b> Defines the maximum size of the read request the DMA Engine can issue to the source interface of the DMA transfer. Supported values are: <ul style="list-style-type: none"> <li>• 4'd7: 128 Bytes</li> <li>• 4'd8: 256 Bytes</li> <li>• 4'd9: 512 Bytes</li> <li>• 4'd10: 1024 Bytes</li> <li>• 4'd11: 2048 Bytes</li> <li>• 4'd12: 4096 Bytes</li> </ul> The DMA Maximum Read Request size should not exceed the QPCIE_UNB_MAXRREQSIZE value in Bytes, however. DMA will restrict its read request size to the maximal read request size of the DMA Source, however. For example, if the DMA Source is an AXI3 interface of 128-bits, the maximal read request size will be limited to 16 beats x 16 bytes = 256 bytes. The higher this value, the better performance will be, but buffer size will also be bigger.
QPCIE_DMA0-7_D_FETCH_N[1:0]	2'd0 - 2'd3	<b>DMA Fetch Descriptor Number:</b> Defines the maximum number of Descriptors that can be fetched by the DMA Engine (0 to 3). SG Descriptor Fetching reduces the latency between the SG Descriptor read request and completion results, so more SG Descriptors result in better performance.

Table 10: Core Constants - DMA Engines Configuration

Constant	Supported Values	Description
QPCIE_DMA0-7_ALIGN_S[1:0]	2'd0 - 2'd3	<b>DMA Source Address Alignment:</b> Defines the start address alignment of the source of the DMA transfer: <ul style="list-style-type: none"> <li>• 0: all DMA transfer start address alignments are supported</li> <li>• 1: DMA transfer start address is aligned on 4-byte boundaries</li> <li>• 2: DMA transfer start address is aligned on the Bridge Data Path (QPCIE_DATAPATH)</li> </ul> Restricting the address alignment reduces the Core gate count and enables higher frequency.
QPCIE_DMA0-7_ALIGN_D[1:0]	2'd0 - 2'd3	<b>DMA Destination Address Alignment:</b> Defines the start address alignment of the destination of the DMA transfer: <ul style="list-style-type: none"> <li>• 0: all DMA transfer start address alignments are supported</li> <li>• 1: DMA transfer start address is aligned on 4-byte boundaries</li> <li>• 2: DMA transfer start address is aligned on Bridge Data Path (QPCIE_DATAPATH)</li> </ul> Restricting the address alignment reduces the Core gate count and enables higher frequency.
QPCIE_DMA0-7_ALIGN_L[1:0]	2'd0 - 2'd3	<b>DMA Length Alignment:</b> Defines the length supported for the DMA transfer: <ul style="list-style-type: none"> <li>• 0: all DMA lengths are supported</li> <li>• 1: DMA length is a multiple of 4 bytes</li> <li>• 2: DMA length is a multiple of the Bridge Data Path (QPCIE_DATAPATH)</li> </ul> Restricting the length reduces the Core gate count and enables higher frequency.
QPCIE_DMA0-7_ALIGN_SD	1'b0 or 1'b1	<b>DMA Source and Destination Relative Alignment:</b> Defines whether the start address of the source and destination of the DMA transfer are aligned. <p>When set to 1'b1, the source and destination start addresses are both aligned with the Bridge Data Path (QPCIE_DATAPATH). For example, the start addresses 0x0013 and 0x0023 are aligned with a Bridge Data Path of 128 bits, but are not aligned with a Bridge Data Path of 256 bits.</p> Restricting address alignment reduces Core gate count and enables higher frequency.
QPCIE_DMA0-7_D_TIMEOUT[2:0]	QPCIE_DMA0-7_D_TIMEOUT[2:0]	<b>DMA Descriptor Polling Timeout:</b> If the next SG-DMA Descriptor is not ready to read or be fetched, the SG-DMA waits for the DMA Descriptor Polling Timeout value specified, before again trying to read the next Descriptor. <p>The DMA Descriptor Polling Timeout is defined in Bridge clock cycles. Supported values are:</p> <ul style="list-style-type: none"> <li>• 3'd0: 512 clock cycles</li> <li>• 3'd1: 1024 clock cycles</li> <li>• ...</li> <li>• 3'd7: 65536 clock cycles</li> </ul>
QPCIE_DMA0-7_PIPE_R_W	QPCIE_DMA0-7_PIPE_R_W	<b>DMA Write RAM Optional Pipeline:</b> When set to 1'b1, a pipeline is implemented on the RAM write port.
QPCIE_DMA0-7_PIPE_U_W	QPCIE_DMA0-7_D_TIMEOUT[2:0]	<b>DMA Write Requests Optional Pipeline:</b> When set to 1'b1, a pipeline is implemented on Write Requests issued by the DMA.

Table 10: Core Constants - DMA Engines Configuration

Constant	Supported Values	Description
<b>DMA Engines Fixed or Dynamic Configuration</b>		
QPCIE_DMA0-7_FIX_SRC	1'b0 or 1'b1	<b>DMA Source Interface Programming:</b> When set to 1'b1, the SRC_ID field of the DMA_SRCPARAM register (see <a href="#">Section 14.5</a> ) is hardwired. When set to 1'b0, it can be dynamically reconfigured in the Bridge Configuration Space.
QPCIE_DMA0-7_SRC_ID[3:0]	/	<b>DMA Source Interface Fixed Value:</b> Specifies the value of the SRC_ID field of the DMA_SRCPARAM register, when the field is hardwired (QPCIE_DMAx_FIX_SRC is 1'b1).
QPCIE_DMA0-7_FIX_DEST	1'b0 or 1'b1	<b>DMA Destination Interface Programming:</b> When set to 1'b1, the DEST_ID field of the DMA_DESTPARAM register (see <a href="#">Section 14.5</a> ) is hardwired. When set to 1'b0, it can be dynamically reconfigured in the Bridge Configuration Space.
QPCIE_DMA0-7_DEST_ID[3:0]	/	<b>DMA Destination Interface Fixed Value:</b> Specifies the value of the DEST_ID field of the DMA_SRCPARAM register, when the field is hardwired (QPCIE_DMA0-7_FIX_DEST is 1'b1).
QPCIE_DMA0-7_FIX_SG	1'b0 or 1'b1	<b>DMA Scatter Gather Parameters Programming:</b> When set to 1'b1, Scatter Gather Support is defined by QPCIE_DMAx_SG_IMPL and the SG_TYPE field of the DMA_CONTROL register (see <a href="#">Section 14.5</a> ) is hardwired. When set to 1'b0, it can be dynamically reconfigured in the Bridge Configuration Space.
QPCIE_DMA0-7_SG_IMPL	1'b0 or 1'b1	<b>DMA Scatter Gather Support Fixed Value:</b> When QPCIE_DMAx_FIX_SG is 1'b1, this constant defines if SG-DMA transfers are supported. <ul style="list-style-type: none"> <li>• If QPCIE_DMAx_FIX_SG is 1'b1 and QPCIE_DMAx_SG_IMPL is 1'b1, both Direct and Scatter Gather DMA transfers are supported.</li> <li>• If QPCIE_DMAx_FIX_SG is 1'b1 and QPCIE_DMAx_SG_IMPL is 1'b0, only Direct DMA transfers are supported.</li> <li>• If QPCIE_DMAx_FIX_SG is 1'b0, or if QPCIE_DMAx_FIX_SG is 1'b1 and QPCIE_DMAx_SG_IMPL is 1'b1, both Direct and Scatter Gather DMA transfers are supported.</li> </ul>
QPCIE_DMA0-7_SG_TYPE[1:0]	/	<b>DMA Scatter Gather Type Fixed Value:</b> Specifies the value of the SG_TYPE field of the DMA_CONTROL register, when the field is hardwired (QPCIE_DMAx_FIX_SG is 1'b1).
QPCIE_DMA0-7_FIX_SG_ID	1'b0 or 1'b1	<b>DMA Scatter Gather Interface Programming:</b> When set to 1'b1, the SG_ID and SG_ID2 fields of the DMA_CONTROL register (see <a href="#">Section 14.5</a> ) are hardwired. When set to 1'b0, they can be dynamically reconfigured in the Bridge Configuration Space.
QPCIE_DMA0-7_SG_ID[2:0]	/	<b>DMA Scatter Gather Interface Fixed Value:</b> Specifies the value of the SG_ID field of the DMA_CONTROL register when the field is hardwired (QPCIE_DMAx_FIX_SG_ID is 1'b1).
QPCIE_DMA0-7_SG_ID2[2:0]	/	<b>DMA Scatter Gather Interface 2 Fixed Value:</b> Specifies the value of the SG2_ID field of the DMA_CONTROL register when the field is hardwired (QPCIE_DMAx_FIX_SG_ID is 1'b1).

Table 10: Core Constants - DMA Engines Configuration

## 5.1.2.8 Address Translation Configuration Core Constants

Constant	Supported Values	Description
<b>Address Translation Static Configuration</b>		
QPCIE_PCIE_WIN0_ENABLE	0 or 1	<b>PCIe Window #0 Implementation:</b> If set to 1, PCI Express Window #0 is implemented.
QPCIE_PCIE_WIN0_SIZE	12 - 64	<b>PCIe Window #0 Size:</b> When PCI Express Window #0 is implemented, its size equals $2^{\text{QPCIE\_PCIE\_WIN0\_SIZE}}$ .
QPCIE_PCIE_WIN1_ENABLE	0 or 1	<b>PCIe Window #1 Implementation:</b> If set to 1, PCI Express Window #1 is implemented.
QPCIE_PCIE_WIN1_SIZE	12 - 64	<b>PCIe Window #1 Size:</b> When PCI Express Window #1 is implemented, its size equals $2^{\text{QPCIE\_PCIE\_WIN1\_SIZE}}$ .
QPCIE_ATR_PCIE_WBUF_NUM	2 - 16	<p><b>PCIe Address Translation Write Buffer Number:</b> This constant defines the number of buffers dedicated to storing PCI Express write requests. Each buffer size is equal to the Bridge Maximum Payload Size. This means that if a write request bigger than the Bridge Maximum Payload Size is received on the PCIe interface, it will be split and stored into several write buffers.</p> <p><b>Note:</b> This constant also defines the maximum number of outstanding write requests that the PCIe Address Translation module can handle.</p> <p>The higher this value, the better performance will be, but buffer size will also be bigger.</p>
QPCIE_ATR_PCIE_WCPL_NUM	2 - 16	<p><b>PCIe Address Translation Write Completion Number:</b> This constant defines the maximum number of outstanding write completion requests that the PCIe Address Translation module can handle.</p> <p>We recommend setting QPCIE_ATR_PCIE_WCPL_NUM equal to QPCIE_ATR_PCIE_WBUF_NUM.</p>
QPCIE_ATR_PCIE_RBUF_NUM	2 - 16	<p><b>PCIe Address Translation Read Buffer Number:</b> This constant defines the number of buffers dedicated to store completions of PCI Express read requests. Each buffer size is equal to the Bridge Maximum Payload Size. This means that if a read request bigger than the Bridge Maximum Payload Size is received on the PCIe interface, the completions will be split and stored into several read buffers.</p> <p><b>Note:</b> This constant also defines the maximum number of outstanding read requests that the PCIe Address Translation module can handle.</p> <p>The higher this value, the better performance will be, but buffer size will also be bigger.</p>
QPCIE_ATR_PCIE_RCPL_NUM	2 - 16	<p><b>PCIe Address Translation Read Completion Number:</b> This constant defines the maximum number of read completion outstanding requests that the PCIe Address Translation module can handle.</p> <p>We recommend setting QPCIE_ATR_PCIE_RCPL_NUM equal to QPCIE_ATR_PCIE_RBUF_NUM.</p>

Table 11: Core Constants - Address Translation Configuration

Constant	Supported Values	Description
QPCIE_ATR_ASLV_WBUF_NUM	2 - 16	<p><b>AXI4 Slave Translation Write Buffer Number:</b> This constant defines the number of buffers dedicated to storing AXI4 Slave write requests. Each buffer size is equal to the Bridge Maximum Payload Size. This means that if a bigger write request is received on the AXI4 Slave interface, it will be split and stored into several write buffers.</p> <p><b>Note:</b> this constant also defines the maximum number of outstanding write requests that the AXI4 Slave Address Translation module can handle.</p> <p>The higher this value, the better performance will be, but buffer size will also be bigger.</p>
QPCIE_ATR_ASLV_WCPL_NUM	2 - 16	<p><b>AXI4 Slave Translation Write Completion Number:</b> This constant defines the maximum number of write completion outstanding requests that the AXI4 Slave Address Translation module can handle.</p> <p>We recommend setting QPCIE_ATR_ASLV_WCPL_NUM equal to QPCIE_ATR_ASLV_WBUF_NUM.</p>
QPCIE_ATR_ASLV_RBUF_NUM	2 - 16	<p><b>AXI4 Slave Translation Read Buffer Number:</b> This constant defines the number of buffers dedicated to storing completions for AXI4 Slave read requests. Each buffer size is equal to the Bridge Maximum Payload Size. This means that if a read request bigger than the Bridge Maximum Payload Size is received on the AXI4 Slave interface, the completions will be split and stored into several read buffers.</p> <p><b>Note:</b> This constant also defines the maximum number of outstanding read requests that the AXI4 Slave Address Translation module can handle.</p> <p>The higher this value, the better performance will be, but buffer size will also be bigger.</p>
QPCIE_ATR_ASLV_RCPL_NUM	2 - 16	<p><b>AXI4 Slave Translation Read Completion Number:</b> This constant defines the maximum number of read completion outstanding requests that the AXI4 Slave Address Translation module can handle.</p> <p>We recommend setting QPCIE_ATR_ASLV_RCPL_NUM equal to QPCIE_ATR_ASLV_RBUF_NUM.</p>
<b>Address Translation Fixed or Dynamic Configuration</b>		
QPCIE_KFIX_ATR_PWIN0-1[7:0]	/	<p><b>PCle Address Translation Table Settings Programming:</b> When QPCIE_KFIX_ATR_PWINi[j] is set to 1'b1, the PCle Window i Address Translation Table Number j registers are hardwired (see <a href="#">Section 14.6</a>). When set to 1'b0, they can be dynamically reconfigured in the Bridge Configuration Space.</p>

Table 11: Core Constants - Address Translation Configuration

Constant	Supported Values	Description
QPCIE_ATR0-7_PWIN0-1_CONF[255:0]	/	<p><b>PCle Address Translation Table Settings:</b> Specifies the value of the PCle Window <i>i</i> Address Translation Table Number <i>j</i> registers when they are hardwired (QPCIE_KFIX_ATR_PWINi[j] is set to 1'b1).</p> <p>QPCIE_ATRj_PWINi_CONF[255:192] is the Translation Table Mask Address. It should be set to (0 - Table Size), where the Table Size is defined by the ATR_SIZE field of the ATR_PARAM register (see <a href="#">Section 14.6</a>). For example, if the Table Size is fixed to 256 KBytes, QPCIE_ATRj_PWINi_CONF[255:192] should be set to (0 - 256KBytes) = 64'hFFFFFFFFFC0000.</p> <p>You can set this mask so that the LSB bits are ignored when the received request address is compared to the table source address to determine if the request matches the translation table.</p>
QPCIE_KFIX_ATR_ASLV0-3[7:0]	/	<p><b>AXI4 Slave Address Translation Table Settings Programming:</b> When QPCIE_KFIX_ATR_ASLVi[j] is set to 1'b1, AXI4 Slave <i>i</i> Address Translation Table Number <i>j</i> registers (see <a href="#">Section 14.6</a>) are hardwired. When set to 1'b0, they can be dynamically reconfigured in the Bridge Configuration Space.</p>
QPCIE_ATR0-7_ASLV0-3_CONF[255:0]	/	<p><b>AXI4 Slave Address Translation Table Settings:</b> Specifies the value of the AXI4 Slave <i>i</i> Address Translation Table Number <i>j</i> registers when they are hardwired (QPCIE_KFIX_ATR_ASLVi[j] is set to 1'b1).</p> <p>QPCIE_ATRj_ASLVi_CONF[255:192] is the Translation Table Mask Address. It should be set to (0 - Table Size), where the Table Size is defined by the ATR_SIZE field of the ATR_PARAM register (see <a href="#">Section 14.6</a>). For example, if the Table Size is fixed to 256 KBytes, QPCIE_ATRj_ASLVi_CONF[255:192] should be set to (0 - 256KBytes) = 64'hFFFFFFFFFC0000.</p> <p>You can set this mask so that the LSB bits are ignored when the received request address is compared to the table source address to determine if the request matches the translation table.</p>

Table 11: Core Constants - Address Translation Configuration

## 5.2 Configuring the Bridge PCI Express Configuration Space

When accessing the Bridge Configuration Space at addresses 0x1000 0x1FFF, the accesses are routed to the PCIe Controller Backend Configuration Space Interface. This enables a Local processor on the AXI domain to read from or write to the PCIe Controller Configuration Space (see [Appendix A: Register Content of the PCIe Config Space](#)).

When several functions are implemented, the PCIE\_CFGNUM register (see [Section 14.1](#)) should be configured appropriately to access the desired function's Configuration Space.

## Chapter 6 Xilinx 7 Series Hard IP Interface

The Core can be connected to the Xilinx 7 Series Integrated Block for PCI Express through the Hard IP interface, whose signals are described in the following table:

Signal	Width	I/O	Description
<b>Clocks and Resets</b>			
<b>sys_clk_p</b>	1	in	Reference clock
<b>sys_clk_n</b>	1	in	Reference clock
<b>sys_rst_n</b>	1	in	Active-low reset signal
<b>Serial Interface to PIPE for internal PHY</b>			
<b>pci_exp_rxn</b>	8	in	Receive input
<b>pci_exp_rxp</b>	8	in	Receive input
<b>pci_exp_txn</b>	8	out	Transmit output
<b>pci_exp_txp</b>	8	out	Transmit output

**Table 12: Hard IP interface signals**



## Chapter 7 Programing a DMA Transfer

The QuickPCIe Expert Core can implement up to eight fully-independent DMA Engine Modules, which enable you to program either Direct DMA transfers or Scatter-Gather DMA (SG-DMA) transfers.

The QuickPCIe Expert DMA Engines allow maximal throughput by implementing:

- a configurable number of outstanding read requests (up to 16)
- configurable buffer size
- configurable descriptor fetching
- completion reordering to allow maximum throughput

The Bridge SG-DMA Engines also provide advanced features to enable maximal flexibility:

- Several scatter-gather types, such as separate Source and Destination SG Chained Lists to enable easy interfacing between two hosts located on different domains (see [Section 7.4](#)).
- Dynamic SG-DMA configuration, such as interrupt generation or start and end criteria for each specific SG page (see [Section 7.3](#) and [Section 7.5](#)).
- Enhanced handshake mechanism between the Host and the Bridge DMA Engine, such as dynamic optional reporting in the descriptor (see [Section 7.6](#)).

### 7.1 Direct and Scatter-Gather DMA Basics

A DMA Transfer includes information about:

- its Source, which is defined by its interface, its transfer parameters (see [Section 7.2](#)) and its starting address.
- its Destination, also defined by its interface, parameters and address.
- the DMA Transfer configuration, which includes the transfer length (up to 4GB, but can be configured as infinite), its controls (start/end/ending criteria/interrupt/reporting request), and its status (error/end status/length).

In Direct DMA transfer mode, the DMA start address is a pointer to a contiguous data buffer mapped in the PCI bus address space. Data is read to and written from the buffer in a sequential order:

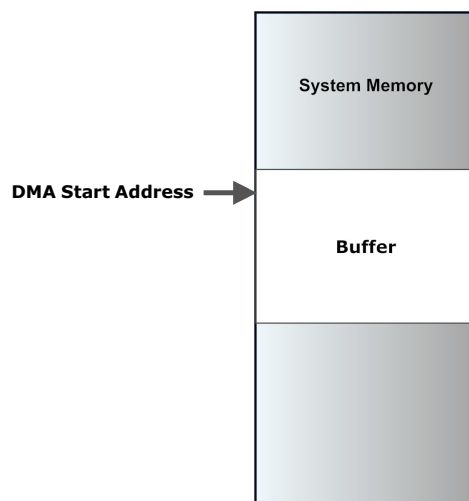


Figure 4: Direct DMA Transfer

In Scatter-Gather transfer mode, the DMA source and/or destination start address is a pointer to a chained list of page descriptors. Each descriptor contains the address and size of a data block (page), as well as a pointer to the next descriptor block to enable circular buffers.

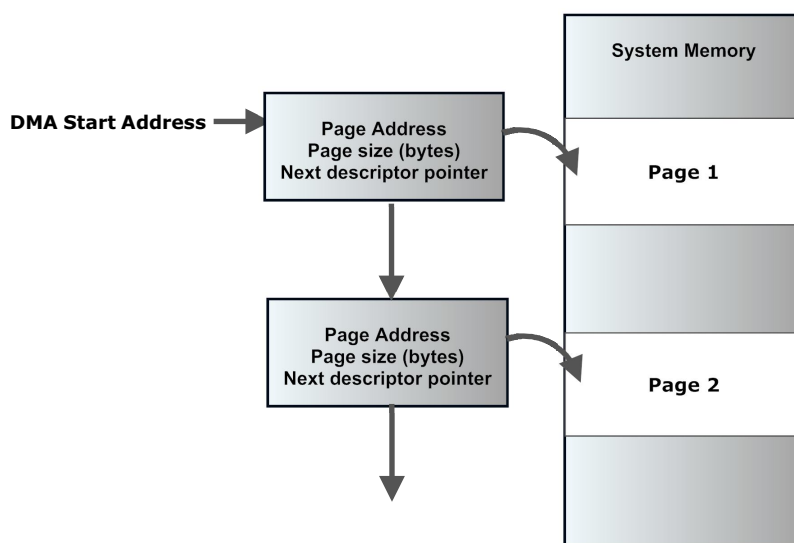


Figure 5: SG-DMA Chained List

## 7.2 Transfer Parameters

The configuration space register TRSF\_PARAM is used to transfer the content of the transfer parameters when using DMA (see [Section 14.5](#)) or Address Translation (see [Section 14.6](#)).

The content of the Transfer Parameters depends on the targeted interface. The following table describes the transfer parameters for each targeted interface:

Targeted Interface	Target ID	Description
PCIe	4'h0	Bit [11]: reserved Bit [10:8]: Traffic Class Bit [7]: ECRC Forward Bit [6]: EP Bit [5:4]: Attr Bit [3]: reserved Bit [2:0]: TLP Type: <ul style="list-style-type: none"> <li>• 3'b000: Memory</li> <li>• 3'b001: Memory Locked</li> <li>• 3'b010: IO</li> <li>• 3'b100: Message</li> </ul>
AXI4-Master	4'h4 4'h7	Bit [11:8]: AQOS Bit [7:5]: APROT Bit [4]: ALOCK Bit [3:0]: ACACHE
AXI4-Stream	4'h8 4'hB	Bit [11:8]: TDEST Bit [7:0]: TID

Table 13: Transfer Parameters

## 7.3 Scatter-Gather Types

The following diagrams illustrate the different types of Scatter-Gather defined for the DMA (DMA\_CONTROL Bits [25:24], see [Section 14.5](#)).

### 7.3.1 SG\_TYPE 00

Independent Scatter-Gather for both Source and Destination.

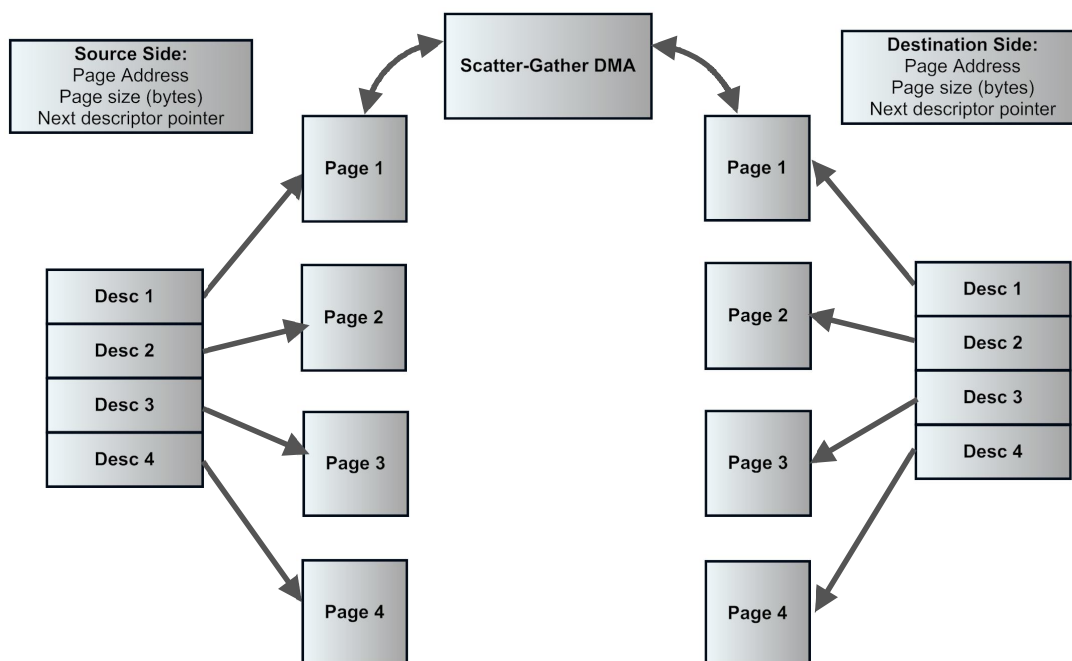


Figure 6: SG\_TYPE 00

### 7.3.2 SG\_TYPE 01

Source address is set according to Descriptor. Destination address is incremented.

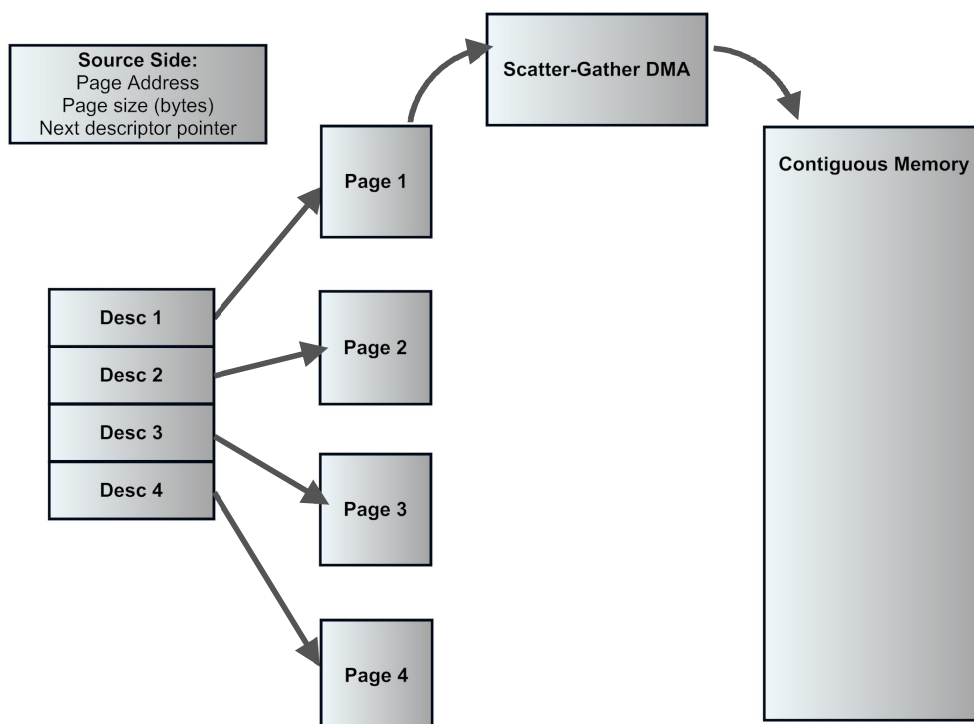


Figure 7: SG\_TYPE 01

### 7.3.3 SG\_TYPE 10

Destination address is set according to Descriptor. Source address is incremented.

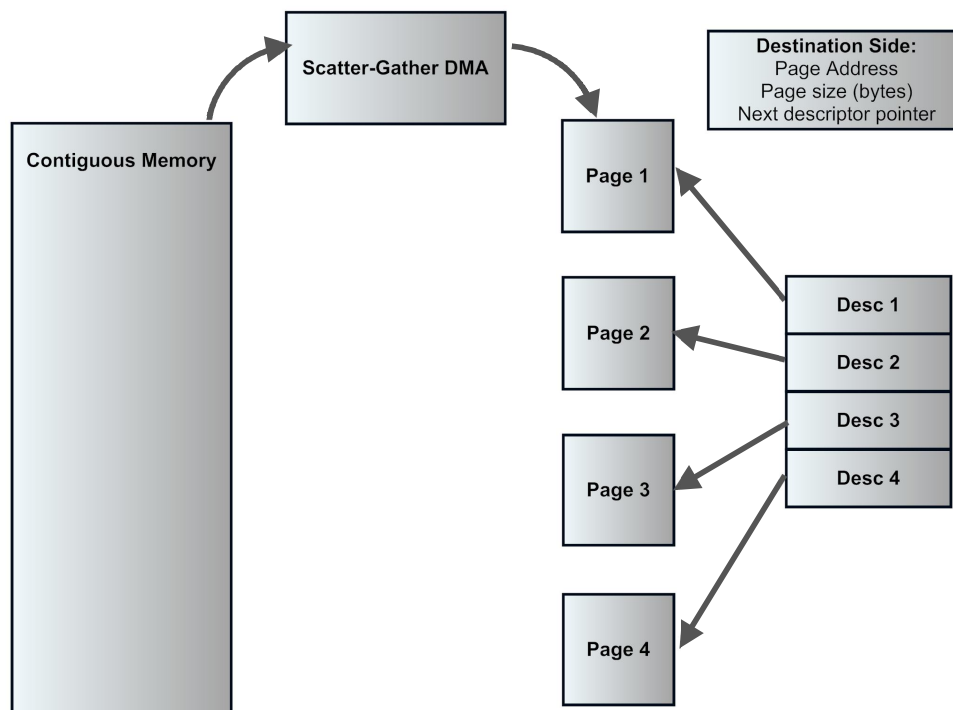


Figure 8: SG\_TYPE 10

### 7.3.4 SG\_TYPE 11

Source and Destination addresses are set according to Descriptor.

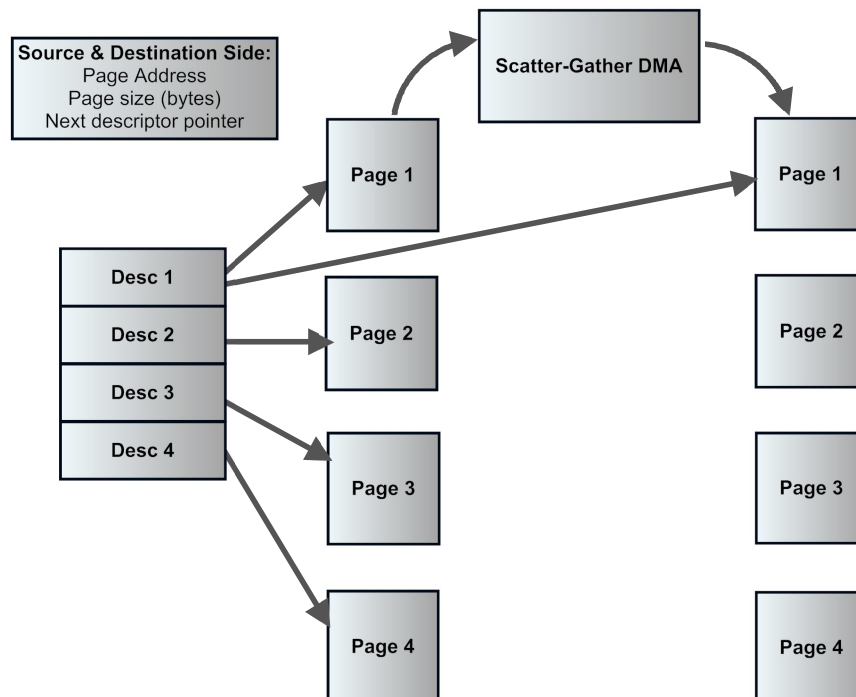


Figure 9: SG\_TYPE 11

## 7.4 Scatter-Gather DMA Descriptors

Scatter-Gather DMA Descriptors are described in the table below.

Name	Byte Offset	R/W	Description
<b>DESC_STATUS</b>	0x00 0x03	RW	<p><b>DESC_STATUS</b> enables dynamic monitoring of the SG-DMA Transfer (when STS_REQ, described below, is set):</p> <ul style="list-style-type: none"> <li>Bits [31:8]: <b>DESC_PRC_PAGE_SIZE</b>: provides the Processed Page Size, which is the actual written or read page size. It can be different from PAGE_SIZE if the Descriptor processing has been shortened due to an error or EOP detection (see SE_COND description).</li> <li>Bits [7:4]: <b>DESC_STATUS</b>: <ul style="list-style-type: none"> <li>Bit 0: SG-DMA Descriptor has been processed.</li> <li>Bit 1: an error occurred during the processing of the current SG-DMA Descriptor.</li> <li>Bit 2: an EOP condition has been reported by the source of the SG-DMA transfer.</li> <li>Bit 3: reserved</li> </ul> </li> </ul> <p><b>Note:</b> It is recommended that the application clears DESC_STATUS, so that it can determine when the STATUS field has been updated by polling Bit 0.</p> <ul style="list-style-type: none"> <li>Bits [3:0]: <b>DESC_STATUS_NUM</b>: provides the Status Number of the DMA Engine. This number is incremented, enabling the application to determine the last processed Descriptor, which is key in streaming flow between asynchronous devices.</li> </ul> <p><b>Note:</b> The DESC_STATUS address offset is equal to 0x00-0x03, to permit the DMA Engine not to read this DWord if it is not required, in order to optimize throughput.</p>
<b>DESC_CONTROL</b>	0x04 0x07	RO	<p><b>DESC_CONTROL</b> enables dynamic control of the SG-DMA Transfer:</p> <ul style="list-style-type: none"> <li>Bits [31:8]: <b>DESC_PAGE_SIZE</b>: provides the Page Size in Bytes, from 1 to 16 Mbytes. If set to 24'h0, it specifies a value of 16 Mbytes.</li> <li>Bits [7:4]: <b>DESC_IRQ</b>: defines when an interrupt should be issued: <ul style="list-style-type: none"> <li>Bit 0: an IRQ is issued when this SG-DMA Descriptor has been processed.</li> <li>Bit 1: an IRQ is issued if an error occurs.</li> <li>Bit 2: an IRQ is issued if the source of the transfer reports an EOP condition.</li> <li>Bit 3: reserved</li> </ul> </li> <li>Bits [3:1]: <b>DESC_TYPE</b>: indicates the current SG-DMA Descriptor mapping. Its value is reserved and equal to 3'b000 for the current SG-DMA Descriptor Mapping.</li> <li>Bit [0]: <b>DESC_STS_REQ</b>: defines whether the DMA Engine provides a status report by writing to DESC_STATUS when the current SG-DMA Descriptor has been processed. This enables the application to easily monitor DMA progress without polling the Bridge's DMA Engine registers.</li> </ul>

**Table 14: Scatter-Gather DMA Descriptors Mapping**

Name	Byte Offset	R/W	Description
DESC_NEXT_ADDR[31:5] DESC_NEXT_RDY DESC_SE_COND	0x08 0x0B	RO	<ul style="list-style-type: none"> <li>• <b>DESC_NEXT_ADDR</b>: Next Descriptor Address. This field must be aligned on a 32-byte boundary.</li> <li>• <b>DESC_NEXT_RDY</b>: indicates if the next SG-DMA Descriptor is ready (and fetchable). The application can set it to 0b to indicate that the chain list is not ended, but will be completed later.</li> <li>• <b>DESC_SE_COND</b>: defines the Start and End conditions for SG-DMA Descriptor processing. It is composed of the following sub-fields: <ul style="list-style-type: none"> <li>• Bit 0: End the DMA transfer after this SG-DMA Descriptor has been processed (equivalent to an End Of Chain).</li> <li>• Bit 1: Abort this SG-DMA Descriptor processing if an error occurs.</li> <li>• Bit 2: <ul style="list-style-type: none"> <li>• If the destination of the SG-DMA is an AXI4 Stream Interface and SG-Type is 01, generate an EOP at the end of SG-DMA Descriptor processing. SE_COND[2] of the DMA_CONTROL parameter must be asserted to enable this feature (see <a href="#">Section 14.5</a>).</li> <li>• If the source of the SG-DMA is an AXI-Stream Interface and SG-Type is 10, stop SG-DMA Descriptor processing if the source of the transfer reports an EOP condition.</li> </ul> </li> <li>• Bit 3: Start this SG-DMA Descriptor processing when the source of the transfer reports an SOP reception (only relevant when the source is an AXI4 Stream Interface, on data following a TLAST assertion).</li> </ul> </li> </ul> <p>If bits [2: 1] of the DESC_SE_COND field are cleared, DMA will stop once the PAGE_SIZE bytes are processed.</p> <p>If the DMA Transfer length is unknown, the application may want to build its SG-DMA Chain List dynamically. In this case, the application can allow the last built SG-DMA Descriptor to be cleared and both the DESC_NEXT_ADDR field and Bit 0 of the DESC_SE_COND field are considered irrelevant for the DMA Engine.</p> <p>Once the application has built the next Descriptor (or if the current SG-DMA Descriptor is the last one in the Chain List), it should update the current SG-DMA Descriptor, setting DESC_NEXT_ADDR and DESC_SE_COND's Bit 0 to their defined values.</p> <p>The DMA Engine regularly checks if the SG-DMA Descriptor has been updated (every 1μs to 65μs, depending on the Core Constant configuration). However the application can set the DESC_UPDT field of the DMA Engine's DMA_CONTROL register to 1b (see <a href="#">Section 14.5</a>) to indicate that a Descriptor has been updated.</p>
NEXT_DESC_ADDR[63:32]	0x0C 0x0F	RO	
DESC_SRC_ADDR[31:0]	0x10 0x13	RO	<b>DESC_SRC_ADDR</b> : Source Address. If not used (SG_TYPE field set to 2'b10), the application can set this field to 64'h0.
DESC_SRC_ADDR[63:32]	0x14 0x17	RO	
DESC_DEST_ADDR[31:0]	0x18 0x1B	RO	<b>DESC_DEST_ADDR</b> : Destination Address. If not used (SG_TYPE field set to 2'b01), the application can set this field to 64'h0.
DESC_DEST_ADDR[63:32]	0x1C 0x1F	RO	

Table 14: Scatter-Gather DMA Descriptors Mapping

## 7.5 Setting Start and End Conditions

The QuickPCIe Expert DMA Engines offer the possibility to configure conditions for the start and end of DMA transfer (for SG-DMA, you can configure these conditions for each individual Descriptor).

These start and end conditions allow the DMA Engines to provide maximum performance in many types of applications.

DMA transfer can be stopped in the following conditions (see the DMA\_CONTROL register description in [Section 14.5](#)):

- If the user clears the Start/Stop bit of the DMA, the DMA is aborted.
- If the DMA is programmed to stop on error and an error occurs, the DMA is aborted.
- In Scatter-Gather DMA transfer mode, if the SG chained list reaches an End Of Chain condition, the DMA is ended.
- If the DMA is programmed to stop on DMA length reached and this occurs, the DMA is ended. If the DMA is not programmed to end on this condition, however, it specifies an infinite length DMA transfer.
- When the source of the DMA transfer is an AXI-Stream interface and if the DMA is programmed to stop on End Of Packet, then the DMA is ended if a TLAST is detected on the AXI-Stream interface.

In Scatter-Gather DMA transfer mode, the Descriptor and its associated Page processing can be stopped on the following per-Descriptor conditions (see the SE\_COND field description in [Section 14.5](#)):

- If the SG Descriptor is configured to stop on error and this condition is reached during its associated page processing, the Descriptor processing is aborted.
- When the source of the DMA transfer is an AXI-Stream interface and if the SG Descriptor is configured to stop on End Of Packet, Descriptor processing is stopped if a TLAST is detected on the AXI-Stream interface.

The Core handles a DMA End command and a DMA Abort command differently.

When the DMA is **aborted**, the transfer is stopped immediately and:

- pending read requests are ignored.
- data stored in the DMA buffers are flushed.
- the reason for the ending of the DMA is reported in the DMA\_STATUS register.
- In SG-DMA mode, fetched descriptors are flushed.

The DMA is effectively stopped either when all pending outstanding requests are completed (this is reported as “DMA successfully stopped”) or after a 256us timeout (reported as “DMA incorrectly ended”). See [Section 14.5](#) for more information.

When the DMA is **ended**, the transfer ends when:

- all required data is sent to the transfer destination.
- all required SG-Descriptor status are reported (see [Section 7.6](#)).
- the reason for the DMA ending is reported in the DMA\_STATUS register.

When Descriptor processing is **aborted**:

- pending read requests related to the Descriptor or its associated page are ignored.
- stored data related to its associated page in the DMA buffers are flushed.
- if SG Reporting is required for the Descriptor, the reasons for the Descriptor processing ending are reported in the DESC\_STATUS field of the Descriptor (see [Section 7.6](#)).

Descriptor processing is effectively ended when all pending outstanding requests are completed.



When a Descriptor processing is **ended**:

- All required data is sent to the transfer destination.
- If SG Reporting is required for this Descriptor, the reasons for the Descriptor processing ending are reported in the DESC\_STATUS field of the Descriptor (see [Section 7.6](#)).

The Descriptor processing is effectively ended when all pending outstanding requests are completed.

## 7.6 Monitoring DMA

The QuickPCIe Expert DMA Engines provides different ways of monitoring DMA transfers, which can be used either independently or conjointly:

- **Through DMA Status Registers:** the DMA\_STATUS and DMA\_PRC\_LENGTH registers provide the current status of the DMA and its processed length (see [Section 14.5](#)).
- **Through Interrupts:** the DMA Engine can be configured to issue interrupts to either the PCIe domain (when the Core is in Endpoint mode) or to the AXI domain for selectable conditions (DMA end, errors detected, End Of Packet detected). In Scatter-Gather mode, an interrupt can be issued to the selected domain on conditions that are configurable per Descriptor (Descriptor processing end, errors detected, End Of Packet detected).
- **Using the SG Reporting feature:** if the STS\_REQ field is set (see [Section 7.4](#)), then at the end of each SG-Descriptor processing, the DMA Engine overwrites the DESC\_STATUS field of the Descriptor, providing information to the software application directly in its memory space.  
Used conjointly with the end condition on End Of Packet reception (see [Section 7.5](#)), it enables packets of unpredictable sizes to be received from the AXI Stream interface, with ultra-low latencies and good performances. Video applications, in particular, can take advantage of the SG Reporting feature, requesting SG Reporting at the end of a frame buffer, and polling on STATUS\_NUM subfield to determine the latest written buffer.

## Chapter 8 Implementing Address Translation

### 8.1 Address Translation Mechanism

The QuickPCIe Expert Core uses Address Translation to:

- convert PCI Express read and write requests to any AXI4 Master interface read and write transaction
- convert any AXI4 Slave interface read and write transaction to PCI Express read and write requests

When the Core is in Endpoint mode, you can implement up to two PCIe Windows.

- When PCIe Window #0 is implemented, PCI Express BAR2/3 is configured as a 64-bit prefetchable memory space whose size is equal to PCIe Window #0. PCIe read and write requests targeting BAR2/3 are then routed to PCIe Window #0 Address Translation module.
- When PCIe Window #1 is implemented, PCI Express BAR4/5 is configured as a 64-bit prefetchable memory space whose size is equal to PCIe Window #1. PCIe read and write requests targeting BAR4/5 are then routed to PCIe Window #1 Address Translation module.

Each Address Translation module can support up to eight Translation Tables.

When transferring PCI Express receive requests to the AXI Master, the Bridge automatically removes the decoded BAR base address, then performs a windows match using the PCIe Window offset address. If a match is found, the Bridge then forwards the request to the desired AXI4 Master Interface and adds the corresponding AXI base address.

For example, in the diagram below, if a write request is received at address 64'hBBBBBBBB\_00020140, the Bridge removes the upper 46-bit address bits, then compares the remaining address 18'h20140 with the PCIe Window #1's tables. The request matches Table #1, and is then forwarded to the AXI4-Master #1 interface, at address 64'hFFFFFFFF\_00000000 + 16'h0140.

The following diagram shows PCI Express to AXI4 Master Address Translation:

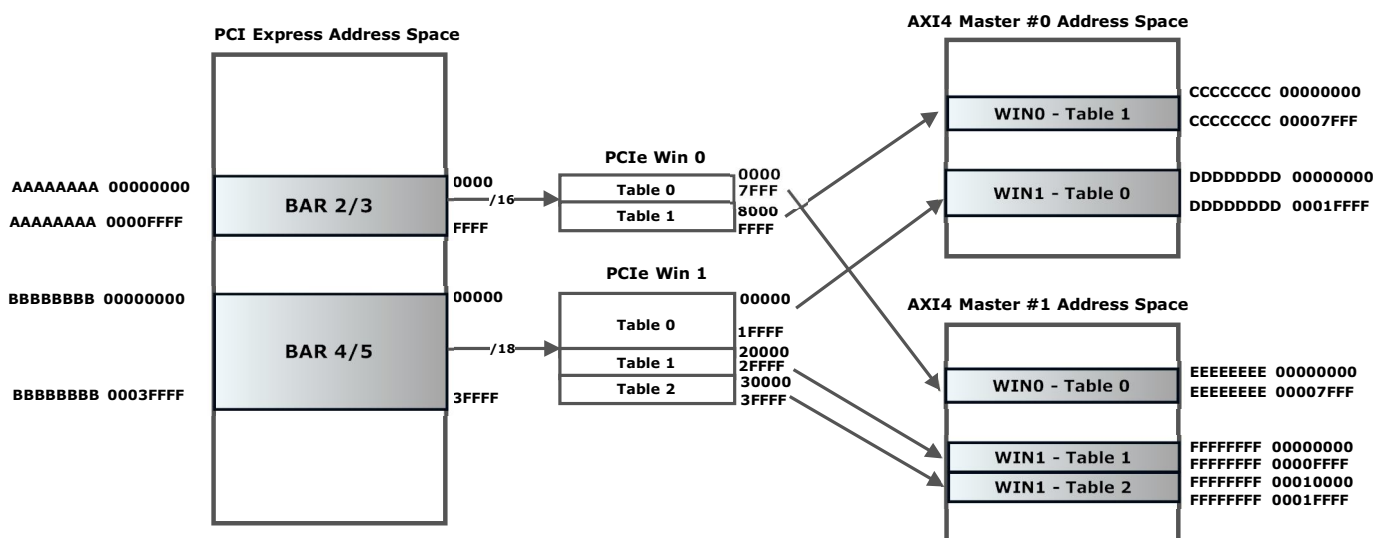


Figure 10: PCIe to AXI4 Master Address Translation

The following diagram shows AXI4 Slave to PCI Express Address Translation:

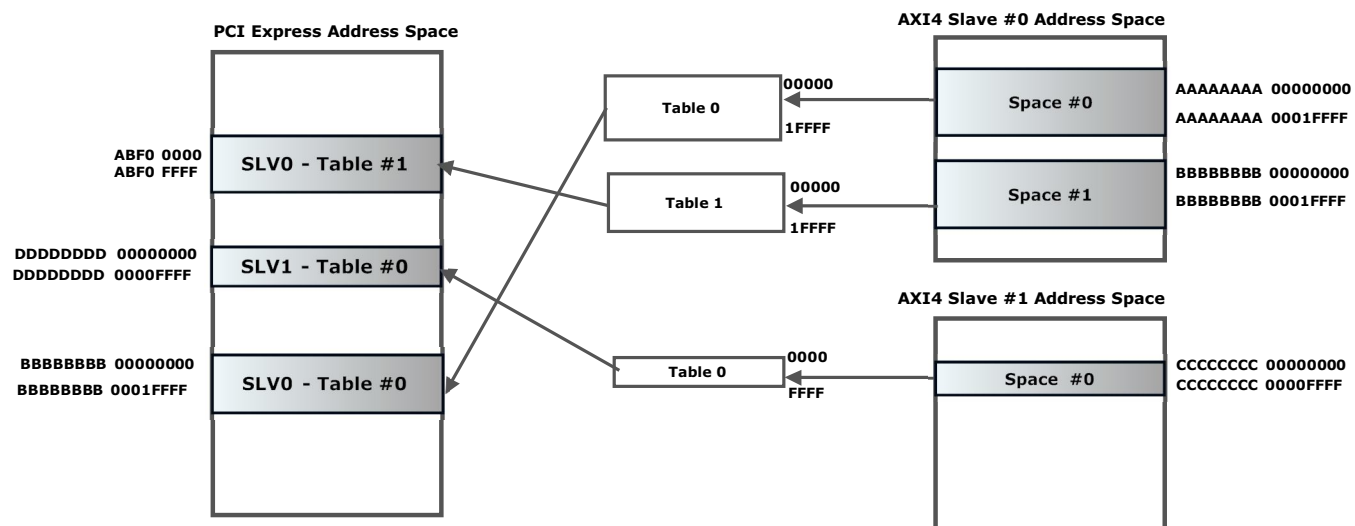


Figure 11: AXI4 Slave to PCIe Address Translation

## 8.2 Setting Address Translation Tables

You can either hardwire each Address Translation Table (for a lower footprint) or set it to be dynamically reconfigurable.

Address Translation settings (see [Section 14.6](#)) enable you to specify:

- whether the Table is implemented or disabled.
- the table size, from 4 KBytes to 16 ExaBytes (that is  $2^{64}$  Bytes). If you set the table size to 16 ExaBytes, no address conversion is performed, as window matching is always successful. You can use this setting if you want to perform address translation outside of the Bridge.
- the source address, which defines the offset address of the table inside the PCIe or AXI Slave Window.
- the Translated interface, to specify on which interface the read or write request should be transferred. It can be set to PCI Express, AXI4-Master, or AXI4-Stream.
- the Transfer parameters, which enable you to define specific PCIe or AXI attributes or parameters per table (see [Section 7.2](#)).

## 8.3 Monitoring Address Translation

The ISTATUS\_X\_ADT\_X registers (see [Section 14.6](#)) enable you to implement monitoring functionality for each Address Translation Table.

The PCI Express Window Address Translation Tables define the following monitoring events:

- **PCIe Doorbell**: reported when a PCIe read or write request targets the Address Translation Table.
- **PCIe Discard Error**: reported to signal a completion timeout on a received PCIe read request.
- **PCIe Fetch Error**: reported to indicate that an error occurred on a PCIe read request (for example, if the corresponding AXI read response reports a SLVERR or a DECERR error).
- **PCIe Post Error**: reported to indicate that an error occurred on a PCIe write request (for example, if the corresponding AXI write response reports a SLVERR or a DECERR error).

The AXI4 Slave Address Translation Tables define the following monitoring events:

- **AXI Doorbell**: reported when a AXI read or write request targets the Address Translation Table.
- **AXI Discard Error**: reported to signal a completion timeout on a received AXI read request.
- **AXI Fetch Error**: reported to indicate that an error occurred on an AXI read request (for example, if the corresponding PCIe read completion has an incorrect ECRC).
- **AXI Post Error**: reported to indicate that an error occurred on an AXI write request.

When one of these events is detected, it is logged to the corresponding ISTATUS\_X\_ADT\_X register, and reported to the corresponding bit of the ISTATUS\_LOCAL and ISTATUS\_HOST registers. If not masked, the event issues an interrupt.

## Chapter 9 AXI Interfaces

### 9.1 AXI4-Lite Slave Interface

The following table list the signals in the AXI4-Lite Slave interface:

Signal	Width	I/O	Description
axi4_slvl_awaddr	14	in	Write address bus
axi4_slvl_awprot	3	in	Protection type
axi4_slvl_awvf	7	in	<b>Virtual Function Number:</b> When virtual functions are implemented, this signal indicates which function has initiated the request. <ul style="list-style-type: none"> <li>• 0 indicates that the physical function has initiated the request,</li> <li>• 1 - 64 indicates that one of the virtual functions 1 to 64 has initiated the request.</li> </ul> This signal must be tied to 0s if virtual functions are not implemented.
axi4_slvl_awvalid	1	in	Write address valid
axi4_slvl_awready	1	out	Write address ready
axi4_slvl_wdata	32	in	Write data
axi4_slvl_wstrb	4	in	Write strobes
axi4_slvl_wvalid	1	in	Write valid
axi4_slvl_wready	1	out	Write ready
axi4_slvl_bresp	2	out	Write response
axi4_slvl_bvalid	1	out	Write response valid
axi4_slvl_bready	1	in	Response ready
axi4_slvl_araddr	14	in	Read address bus
axi4_slvl_arprot	3	in	Protection type
axi4_slvl_arvf	7	in	<b>Virtual Function Number:</b> When virtual functions are implemented, this signal indicates which function has initiated the request. <ul style="list-style-type: none"> <li>• 0 indicates that the physical function has initiated the request,</li> <li>• 1 - 64 indicates that one of the virtual functions 1 to 64 has initiated the request.</li> </ul> This signal must be tied to 0s if virtual functions are not implemented.
axi4_slvl_arvalid	1	in	Read address valid
axi4_slvl_arready	1	out	Read address ready
axi4_slvl_rdata	32	out	Read data
axi4_slvl_rresp	2	out	Read response
axi4_slvl_rvalid	1	out	Read valid
axi4_slvl_rready	1	in	Read ready

Table 15: AXI4-Lite Slave interface signals

## 9.2 AXI4-Lite Master Interface

The following table list the signals in the AXI4-Lite Master interface:

Signal	Width	I/O	Description
axi4_mstl_awaddr	13	out	Write address bus
axi4_mstl_awprot	3	out	Protection type
axi4_mstl_awvf	7	out	<b>Virtual Function Number:</b> When virtual functions are implemented, this signal indicates which function has initiated the request. <ul style="list-style-type: none"> <li>• 0 indicates that the physical function has initiated the request,</li> <li>• 1 - 64 indicates that one of the virtual functions 1 to 64 has initiated the request.</li> </ul> This signal must be tied to 0s if virtual functions are not implemented.
axi4_mstl_awvalid	1	out	Write address valid
axi4_mstl_awready	1	in	Write address ready
axi4_mstl_wdata	32	out	Write data
axi4_mstl_wstrb	4	out	Write strobes
axi4_mstl_wvalid	1	out	Write valid
axi4_mstl_wready	1	in	Write ready
axi4_mstl_bresp	2	in	Write response
axi4_mstl_bvalid	1	in	Write response valid
axi4_mstl_bready	1	out	Response ready
axi4_mstl_araddr	13	out	Read address bus
axi4_mstl_arprot	3	out	Protection type
axi4_mstl_arvf	7	out	<b>Virtual Function Number:</b> When virtual functions are implemented, this signal indicates which function has initiated the request. <ul style="list-style-type: none"> <li>• 0 indicates that the physical function has initiated the request,</li> <li>• 1 - 64 indicates that one of the virtual functions 1 to 64 has initiated the request.</li> </ul> This signal must be tied to 0s if virtual functions are not implemented.
axi4_mstl_arvalid	1	out	Read address valid
axi4_mstl_arready	1	in	Read address ready
axi4_mstl_rdata	32	in	Read data
axi4_mstl_rresp	2	in	Read response
axi4_mstl_rvalid	1	in	Read valid
axi4_mstl_rready	1	out	Read ready

Table 16: AXI4-Lite Master interface signals

## 9.3 AXI4 Master Descriptor Interface

The following table list the signals in the AXI4 Master Descriptor interface:

Signal	Width	I/O	Description
axi4_mstd_awid	4	out	Write address ID
axi4_mstd_awaddr	64	out	Write address bus
axi4_mstd_awregion	4	out	Write region
axi4_mstd_awlen	8	out	Burst length
axi4_mstd_awsz	3	out	Burst size
axi4_mstd_awburst	2	out	Burst type
axi4_mstd_awlock	1	out	Lock type
axi4_mstd_awcache	4	out	Cache type
axi4_mstd_awprot	3	out	Protection type
axi4_mstd_awqos	4	out	QoS value
axi4_mstd_awvalid	1	out	Write address valid
axi4_mstd_awready	1	in	Write address ready
axi4_mstd_wid	4	out	Write ID (for AXI3; unused in AXI4)
axi4_mstd_wdata	64, 128, or 256	out	Write data
axi4_mstd_wstrb	8, 16, or 32	out	Write strobes
axi4_mstd_wlast	1	out	Write last
axi4_mstd_wvalid	1	out	Write valid
axi4_mstd_wready	1	in	Write ready
axi4_mstd_bid	4	in	Response ID
axi4_mstd_bresp	2	in	Write response
axi4_mstd_bvalid	1	in	Write response valid
axi4_mstd_bready	1	out	Response ready
axi4_mstd_arid	4	out	Read address ID
axi4_mstd_araddr	64	out	Read address bus
axi4_mstd_arregion	4	out	Read region
axi4_mstd_arlen	8	out	Burst length
axi4_mstd_arsz	3	out	Burst size
axi4_mstd_arburst	2	out	Burst type
axi4_mstd_arlock	1	out	Lock type
axi4_mstd_arcache	4	out	Cache type
axi4_mstd_arprot	3	out	Protection type
axi4_mstd_arqos	4	out	QoS value
axi4_mstd_arvalid	1	out	Read address valid

Table 17: AXI4 Master Descriptor interface signals

Signal	Width	I/O	Description
axi4_mstd_arready	1	in	Read address ready
axi4_mstd_rid	4	in	Response ID
axi4_mstd_rdata	64, 128, or 256	in	Read data
axi4_mstd_rresp	2	in	Read response
axi4_mstd_rlast	1	in	Read last
axi4_mstd_rvalid	1	in	Read valid
axi4_mstd_rready	1	out	Read ready

Table 17: AXI4 Master Descriptor interface signals



## 9.4 AXI4 Master Interfaces (0 - 3)

The following table list the signals in the AXI4 Master 0 interface. The same signals are used for the AXI4 Master 1 (axi4\_mst1\_), AXI4 Master 2 (axi4\_mst2\_), and AXI4 Master 3 (axi4\_mst3\_) interface signals.

Signal	Width	I/O	Description
axi4_mst0_awid	4	out	Write address ID
axi4_mst0_awaddr	64	out	Write address bus
axi4_mst0_awregion	4	out	Write region
axi4_mst0_awlen	8	out	Burst length
axi4_mst0_awsz	3	out	Burst size
axi4_mst0_awburst	2	out	Burst type
axi4_mst0_awlock	1	out	Lock type
axi4_mst0_awcache	4	out	Cache type
axi4_mst0_awprot	3	out	Protection type
axi4_mst0_awqos	4	out	QoS value
axi4_mst0_awuser	32	out	<b>Write Address user side band signal:</b> <ul style="list-style-type: none"> <li>• Bit [0]: asserted to 1b when an ECRC error is detected in the AXI Burst</li> <li>• Bit [1]: asserted to 1b when a PCIe Memory Error (Poisoned TLP Reception or Receive Buffer Memory Error) is detected in the AXI Burst</li> <li>• Bit [2]: asserted to 1b when a Bridge Buffer Memory Error is detected in the AXI Burst</li> <li>• Bit [7:3]: reserved</li> <li>• Bit [9:8]: Address Type (00b: Untranslated, 01b: Translation Request, 10b: Translated)</li> <li>• Bit [12:10]: reserved</li> <li>• Bit [13]: PCIe Attribute value for NoSnoop</li> <li>• Bit [14]: PCIe Attribute value for Relaxed Ordering</li> <li>• Bit [15]: PCIe Attribute value for ID-Based Ordering</li> <li>• Bit [31:16]: PCIe Requester ID</li> </ul>
axi4_mst0_awvalid	1	out	Write address valid
axi4_mst0_awready	1	in	Write address ready
axi4_mst0_wid	4	out	Write ID (for AXI3; unused in AXI4)
axi4_mst0_wdata	64, 128, or 256	out	Write data
axi4_mst0_wderr	8, 16, or 32	out	<b>Write Data Error:</b> <ul style="list-style-type: none"> <li>• Bit [0]: Write data memory error. This error is reported by the Bridge in the data phase(s) in which it was detected.</li> <li>• Bit [31:1]: reserved.</li> </ul>
axi4_mst0_wstrb	8, 16, or 32	out	Write strobes
axi4_mst0_wlast	1	out	Write last
axi4_mst0_wvalid	1	out	Write valid
axi4_mst0_wready	1	in	Write ready
axi4_mst0_bid	4	in	Response ID

Table 18: AXI4 Master 0 interface signals

Signal	Width	I/O	Description
axi4_mst0_bresp	2	in	Write response
axi4_mst0_bvalid	1	in	Write response valid
axi4_mst0_bready	1	out	Response ready
axi4_mst0_arid	4	out	Read address ID
axi4_mst0_araddr	64	out	Read address bus
axi4_mst0_arregion	4	out	Read region
axi4_mst0_arlen	8	out	Burst length
axi4_mst0_arsize	3	out	Burst size
axi4_mst0_arburst	2	out	Burst type
axi4_mst0_arlock	1	out	Lock type
axi4_mst0_arcache	4	out	Cache type
axi4_mst0_arprot	3	out	Protection type
axi4_mst0_arqos	4	out	QoS value
axi4_mst0_aruser	32	out	<b>Read Address user side band signal:</b> <ul style="list-style-type: none"> <li>• Bit [0]: asserted to 1b when an ECRC error is detected in the AXI Burst</li> <li>• Bit [1]: asserted to 1b when a PCIe Memory Error (Poisoned TLP Reception or Receive Buffer Memory Error) is detected in the AXI Burst</li> <li>• Bit [2]: asserted to 1b when a Bridge Buffer Memory Error is detected in the AXI Burst</li> <li>• Bit [7:3]: reserved</li> <li>• Bit [9:8]: Address Type (00b: Untranslated, 01b: Translation Request, 10b: Translated)</li> <li>• Bit [10]: Translation Request No Write (NW) flag</li> <li>• Bit [12:11]: reserved</li> <li>• Bit [13]: PCIe Attribute value for NoSnoop</li> <li>• Bit [14]: PCIe Attribute value for Relaxed Ordering</li> <li>• Bit [15]: PCIe Attribute value for ID-Based Ordering</li> <li>• Bit [31:16]: PCIe Requester ID</li> </ul>
axi4_mst0_arvalid	1	out	Read address valid
axi4_mst0_arready	1	in	Read address ready
axi4_mst0_rid	4	in	Response ID
axi4_mst0_rdata	64, 128, or 256	in	Read data
axi4_mst0_r derr	8, 16, or 32	in	<b>Read Data Error:</b> <ul style="list-style-type: none"> <li>• Bit [0]: Read data error. This error allows the Application to report a data error in the data phase(s) in which it is asserted.</li> <li>• Bit [31:1]: reserved.</li> </ul>
axi4_mst0_rresp	2	in	Read response
axi4_mst0_rlast	1	in	Read last
axi4_mst0_rvalid	1	in	Read valid
axi4_mst0_rready	1	out	Read ready

Table 18: AXI4 Master 0 interface signals

## 9.5 AXI4 Slave Interfaces (0 - 3)

The following table list the signals in the AXI4 Slave 0 interface. The same signals are used for the AXI4 Slave 1 (axi4\_slv1\_), AXI4 Slave 2 (axi4\_slv2\_), and AXI4 Slave 3 (axi4\_slv3\_) interface signals.

Signal	Width	I/O	Description
axi4_slv0_awid	4	in	Write address ID
axi4_slv0_awaddr	64	in	Write address bus
axi4_slv0_awregion	4	in	Write region
axi4_slv0_awlen	8	in	Burst length
axi4_slv0_awsz	3	in	Burst size
axi4_slv0_awburst	2	in	Burst type
axi4_slv0_awlock	1	in	Lock type
axi4_slv0_awcache	4	in	Cache type
axi4_slv0_awprot	3	in	Protection type
axi4_slv0_awqos	4	in	QoS value
axi4_slv0_awvalid	1	in	Write address valid
axi4_slv0_awready	1	out	Write address ready
axi4_slv0_wid	4	in	Write ID (for AXI3; unused in AXI4)
axi4_slv0_wdata	64, 128, or 256	in	Write data
axi4_slv0_wderr	8, 16, or 32	in	<b>Write Data Error:</b> <ul style="list-style-type: none"> <li>• Bit [0]: Write data memory error. This error allows the Application to report a data error in the data phase(s) in which it is asserted.</li> <li>• Bit [31:1]: reserved.</li> </ul>
axi4_slv0_wstrb	8, 16, or 32	in	Write strobes
axi4_slv0_wlast	1	in	Write last
axi4_slv0_wvalid	1	in	Write valid
axi4_slv0_wready	1	out	Write ready
axi4_slv0_bid	4	out	Response ID
axi4_slv0_bresp	2	out	Write response
axi4_slv0_bvalid	1	out	Write response valid
axi4_slv0_bready	1	in	Response ready
axi4_slv0_arid	4	in	Read address ID
axi4_slv0_araddr	64	in	Read address bus
axi4_slv0_arregion	4	in	Read region
axi4_slv0_arlen	8	in	Burst length
axi4_slv0_arsz	3	in	Burst size
axi4_slv0_arburst	2	in	Burst type
axi4_slv0_arlock	1	in	Lock type

Table 19: AXI4 Slave 0 interface signals

Signal	Width	I/O	Description
axi4_slv0_arcache	4	in	Cache type
axi4_slv0_arprot	3	in	Protection type
axi4_slv0_argos	4	in	QoS value
axi4_slv0_arvalid	1	in	Read address valid
axi4_slv0_arready	1	out	Read address ready
axi4_slv0_rid	4	out	Response ID
axi4_slv0_rdata	64, 128, or 256	out	Read data
axi4_slv0_r derr	8, 16, or 32	out	<b>Read Data Error:</b> <ul style="list-style-type: none"> <li>• Bit [0]: Read data error. This error is reported by the Bridge in the data phase(s) in which it was detected.</li> <li>• Bit [31:1]: reserved.</li> </ul>
axi4_slv0_rresp	2	out	Read response
axi4_slv0_rlast	1	out	Read last
axi4_slv0_ruser	32	out	<b>Read user side band signal:</b> <ul style="list-style-type: none"> <li>• Bit [0]: asserted to 1b when an ECRC error is detected in the AXI Burst</li> <li>• Bit [1]: asserted to 1b when a PCIe Memory Error (Poisoned TLP Reception or Receive Buffer Memory Error) is detected in the AXI Burst</li> <li>• Bit [2]: asserted to 1b when a Bridge Buffer Memory Error is detected in the AXI Burst</li> <li>• Bit [12:3]: reserved</li> <li>• Bit [13]: PCIe Attribute value for NoSnoop</li> <li>• Bit [14]: PCIe Attribute value for Relaxed Ordering</li> <li>• Bit [15]: PCIe Attribute value for ID-Based Ordering</li> <li>• Bit [31:16]: PCIe Completer ID</li> </ul>
axi4_slv0_rvalid	1	out	Read valid
axi4_slv0_rready	1	in	Read ready

Table 19: AXI4 Slave 0 interface signals

## 9.6 AXI4 Stream Interfaces (0 - 3)

The following table list the signals in the AXI4 Stream 0 interface. The same signals are used for the AXI4 Stream 1 (axi4\_sto1\_/axi4\_sti1), AXI4 Stream 2 (axi4\_sto2\_/axi4\_sti2), and AXI4 Stream 3 (axi4\_sto3\_/axi4\_sti3) interface signals.

Signal	Width	I/O	Description
<b>AXI4 Stream Out Channel</b>			
axi4_sto0_tvalid	1	out	Indicates that the Master is driving a valid transfer.
axi4_sto0_tready	1	in	Indicates that a slave can accept a transfer in the current cycle.
axi4_sto0_tdata	64, 128, or 256	out	Primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.
axi4_sto0_tstrb	8, 16, or 32	out	Byte qualifier that indicates whether the content of the associated byte of TDATA is processed as a data byte or a position byte.
axi4_sto0_tkeep	64, 128, or 256	out	Byte qualifier that indicates whether the content of the associated byte of TDATA is processed as part of the data stream.
axi4_sto0_tlast	1	out	Indicates the boundary of a packet.
axi4_sto0_tid	8	out	Data stream identifier
axi4_sto0_tdest	4	out	Provides routing information for the data stream.
<b>AXI4 Stream In Channel</b>			
axi4_sti0_tvalid	1	in	Indicates that the Master is driving a valid transfer.
axi4_sti0_tready	1	out	Indicates that a slave can accept a transfer in the current cycle.
axi4_sti0_tdata	64, 128, or 256	in	Primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.
axi4_sti0_tstrb	8, 16, or 32	in	Byte qualifier that indicates whether the content of the associated byte of TDATA is processed as a data byte or a position byte.
axi4_sti0_tkeep	8, 16, or 32	in	Byte qualifier that indicates whether the content of the associated byte of TDATA is processed as part of the data stream.
axi4_sti0_tlast	1	in	Indicates the boundary of a packet.
axi4_sti0_tid	8	in	Data stream identifier
axi4_sti0_tdest	4	in	Provides routing information for the data stream.

**Table 20: AXI4 Stream interface signals**

## 9.7 AXI4 Read and Write Burst Examples

This section provides examples of basic AXI protocol transactions. Each example shows the `VALID` and `READY` handshake mechanism. Transfer of either address information or data occurs when both the `VALID` and `READY` signals are high.

The following diagram shows a read burst of four transfers. The Master drives the address and the Slave accepts it one cycle later.

After the address appears on the address bus, the data transfer occurs on the read data channel. The slave keeps the `VALID` signal low until the read data is available. For the final data transfer of the burst, the slave asserts the `RLAST` signal to show that the last data item is being transferred.

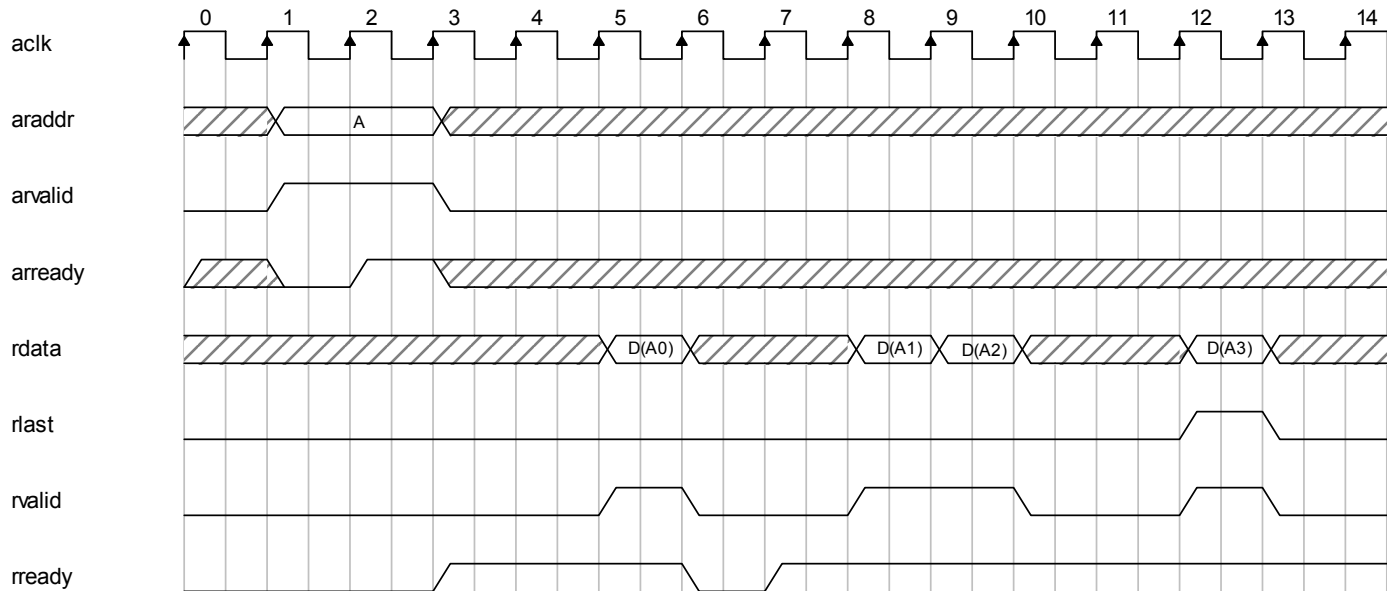


Figure 12: Read Burst Example 1

The diagram below shows how a master can drive another burst address after the slave accepts the first address. This enables a slave to begin processing data for the second burst in parallel with the completion of the first burst.

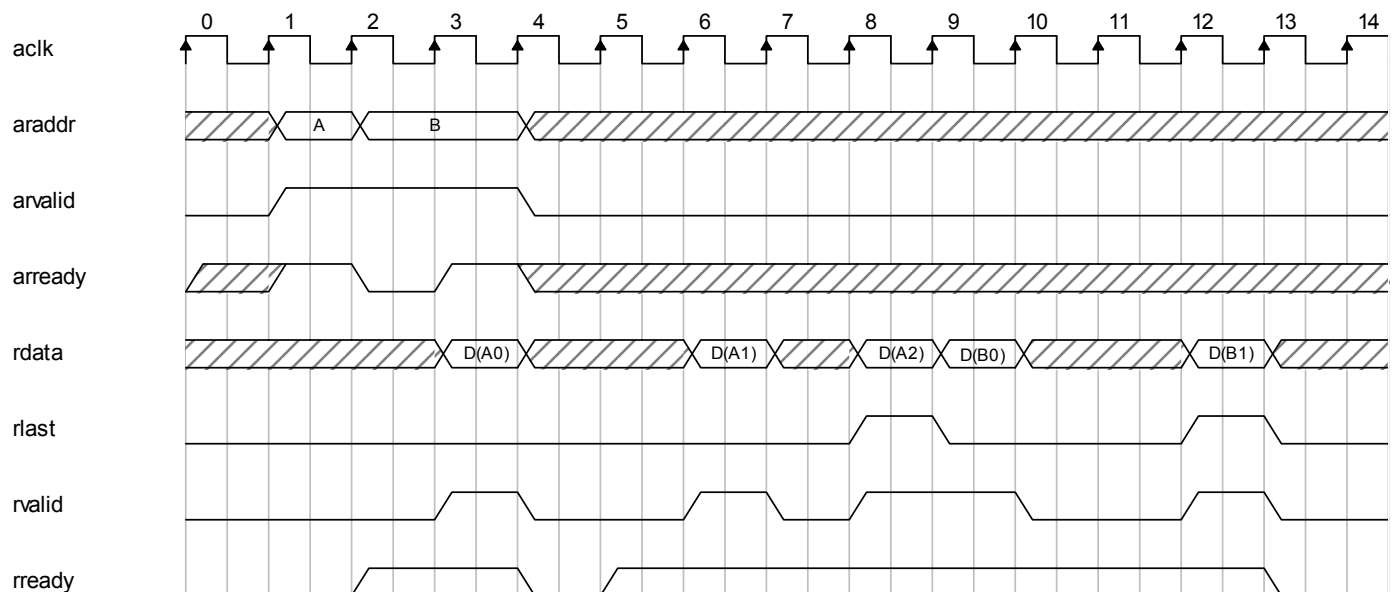


Figure 13: Read Burst Example 2

The following diagram shows a write transaction. The process starts when the master sends an address and control information on the write address channel. The master then sends each item of write data over the write data channel. When the master sends the last data item, the WLAST signal is driven high. When the slave has accepted all the data items, it drives a write response back to the master to indicate that the write transaction is complete.

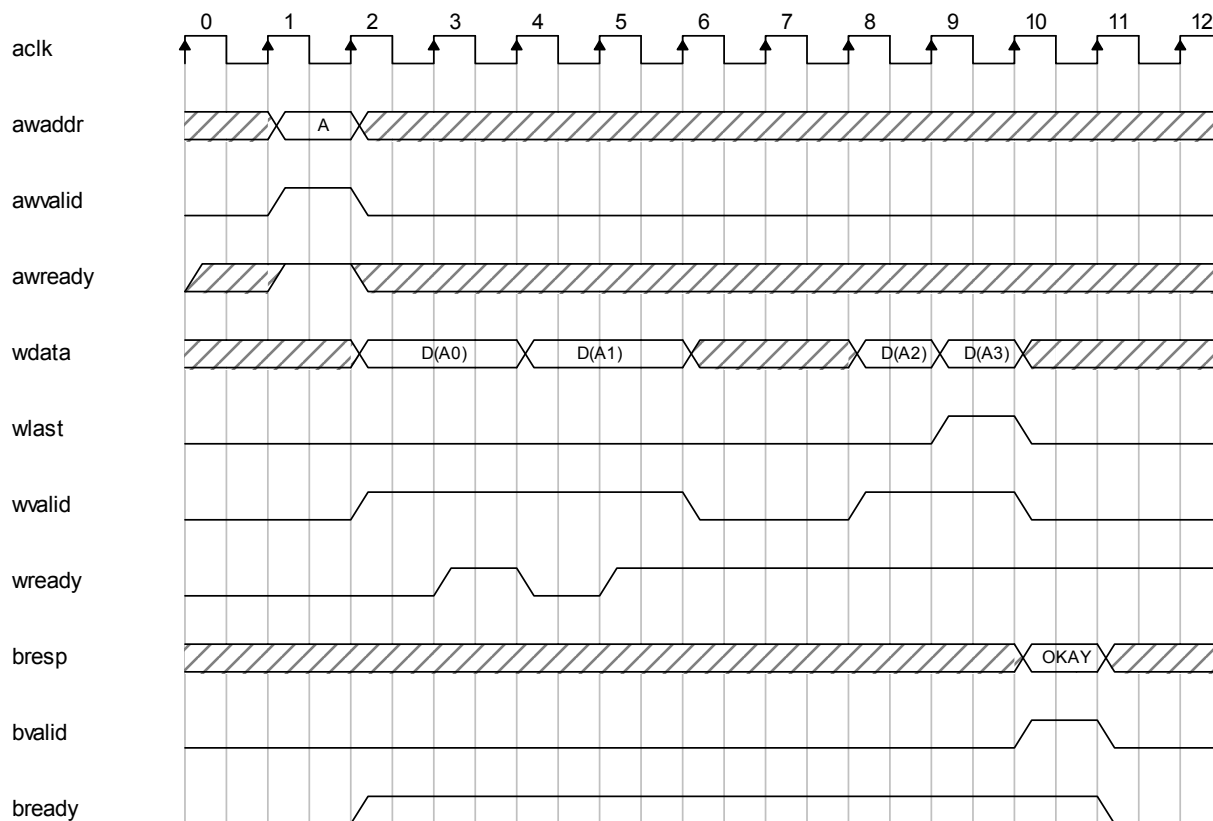


Figure 14: Write Burst Example

## Chapter 10 Handling Interrupts

### 10.1 Handling Host Interrupts

The QuickPCIe Expert Core can issue interrupts to the Host processor (on the PCIe domain) for the following events:

- DMA transfer end or error
- Address Translation doorbell or error
- Local interrupts

When one of these events occurs, the source of the interrupt is reported in the ISTATUS\_HOST register.

The Host processor can mask or enable each interrupt source independently by setting or clearing the corresponding bit in the IMASK\_HOST register.

See [Section 14.2](#) for more information about these registers.

When an interrupt source is active and not masked, an interrupt is issued on the PCIe domain. Processing of this interrupt when received by the Host processor is application-specific. In general, however, the Host:

- Reads the ISTATUS\_HOST register to determine the source of the interrupt.
- Reads any other Bridge Configuration Space status registers required.
- Reads any External status registers required.
- Performs the requisite actions.
- Clears the ISTATUS\_HOST interrupt source by writing 1 to the corresponding bit.

### 10.2 Handling Local Interrupts

The QuickPCIe Expert Core can issue interrupts to the Local processor (on the AXI domain) for the following events:

- DMA transfer end or error
- Address Translation doorbell or error
- Legacy power management state change

When one of these events occurs, the source of the interrupt is reported in bits 31 to 24 of the ISTATUS\_LOCAL register.

The Host processor can mask or enable each interrupt source independently by setting or clearing the corresponding bit in the IMASK\_LOCAL register. See [Section 14.2](#) for more information about these registers.

When an interrupt source is active and not masked, an interrupt is issued on the AXI domain and reported through the QPCIE\_INTERRUPT\_OUT[31:0] output port.

Processing of this interrupt when received by the Local processor is application-specific. In general, however, the Local processor:

- Reads the ISTATUS\_LOCAL register to determine the source of the interrupt.
- Reads any other Bridge Configuration Space status registers required.
- Reads any PCI Express configuration space registers required.
- Reads any PCI Express device status registers required.
- Performs the requisite actions.
- Clears the ISTATUS\_LOCAL interrupt source by writing 1 to the corresponding bit.



The following table describes the QuickPCIe Expert local interrupt signals. Note that both signals are synchronous to the Transaction Layer Clock (QPCIE\_CLK).

Signal	I/O	Description
qpcie_interrupt_in[7:0]	in	<b>Input to the QuickPCIe Expert:</b> The local processor can drive up to 8 interrupt sources by generating a pulse (high) on the QPCIE_INTERRUPT_IN [7 : 0] input port. The method used by the host processor to identify the source of the interrupt is user-defined (by reading a specific External Register's address, for example).
qpcie_interrupt_out[31:0]	out	<b>Output to the QuickPCIe Expert:</b> The local processor on the AXI domain can receive up to 32 interrupt sources. This signal is equivalent to a logical AND of the ISTATUS_LOCAL and IMASK_LOCAL registers; it can be used to monitor each individual interrupt source without reading the ISTATUS_LOCAL register.

**Table 21: Local interrupt signals**

When the QPCIE\_EXPOSED\_INTERRUPT core constant is set to 1 (see [Section 5.1.2.4](#)), the behavior of LOCAL\_INTERRUPT\_OUT [31 : 24] is modified as follows:

- **local\_interrupt\_out[31]:** System Error (Rootport only): This signal is asserted for one clock cycle when a system error, as defined by the *PCI Express Specifications*, occurs.
- **local\_interrupt\_out[30]:** Power Management or Hotplug Event interrupt (Rootport only): This signal is asserted when a power management or hotplug event occurs, and remains asserted until the interrupt source register has been cleared.
- **local\_interrupt\_out[29]:** AER Error (Rootport only): This signal is asserted for one clock cycle when an AER error occurs, and remains asserted until the interrupt source register has been cleared.
- **local\_interrupt\_out[28]:** MSI received: Asserted if an MSI has been captured at IMSI\_ADDR in the Bridge Configuration Space.
- **local\_interrupt\_out[27]:** Interrupt Line State: Asserted when PCI interrupt line D is asserted.
- **local\_interrupt\_out[26]:** Interrupt Line State: Asserted when PCI interrupt line C is asserted.
- **local\_interrupt\_out[25]:** Interrupt Line State: Asserted when PCI interrupt line B is asserted.
- **local\_interrupt\_out[24]:** Interrupt Line State: Asserted when PCI interrupt line A is asserted.

## 10.3 Implementing MSI

Message Signaled Interrupts (MSI) enable the PCI Express device to deliver interrupts by performing memory write transactions, instead of using the INTx message emulation mechanism.

On power up, the Core starts in INTx mode. To use MSI instead of INTx messages, the Host should set the MSI capability structure in all the PCI Express devices that will use MSI.

MSI message address is 64-bits and should be set equal to the QuickPCIe Expert MSI capture address.

In Endpoint mode, MSI can only be generated; they can not be received.

If the Core is configured to support MSI, it is the responsibility of the PCIe Host to set MSI by programming the corresponding control register of the PCI Express controller configuration space.

When 32 MSI are implemented, the data carried by the MSI includes the number(s) of the interrupt source(s).

## Chapter 11 Managing Low Power

The following sections describe the various low power states and how these states are handled by the Core.

### 11.1 ASPM L0s Low Power

This low-power state does not require any application action; it is handled automatically by the Core.

- The Core enters ASPM L0s after a pre-defined period of PCIe transmit inactivity (set by the ASPM L0s entry delay defined by the PCIE\_PEX\_SPC2 register).
- It exits ASPM L0s as soon as a packet needs to be transmitted to the PCIe.

### 11.2 ASPM L1 Low Power

This low-power state does not require any application action; it is handled automatically by the Core.

- The Core enters ASPM L1 after a pre-defined period of inactivity (set by the ASPM L1 entry delay defined by the PCIE\_PEX\_SPC2 register).
- It exits ASPM L1 as soon as a packet needs to be transmitted to the PCIe or when its link partner exits this state.

### 11.3 L1 Low Power

Entry to the L1 low-power state can only be initiated by the Core when its legacy low-power state is not D0.

This low-power state does not require any application action; it is handled automatically by the Core.

- The Core enters L1 when the legacy low-power state is set to D1, D2, or D3 by the host (the legacy power state can be checked in the ISTATUS\_PM register; any change is reported by the PM\_MSI\_INT of the ISTATUS\_LOCAL register) and there are no more packets to be transmitted.
- It exits L1 when directed to do so by its link partner.

### 11.4 Handling Power Management Events

The Core can send a power management event to the host when it is in a low-power state and needs to be restored to a fully-functional power state. This occurs when the application requests a power management event by setting the 'Send PME' bit of the ICMD\_PM register.

## Chapter 12 Test Interface

The following table describes the QuickPCIe Expert test interface, which is synchronous to the Transaction Layer Clock (QPCIE\_CLK).

Signal	I/O	
test_out[63:0]	out	<ul style="list-style-type: none"> <li>• Bit [5:0] <b>Negotiated Link Width</b>: indicates the negotiated width of the given PCI Express Link: <ul style="list-style-type: none"> <li>• 00 0001b: x1</li> <li>• 00 0010b: x2</li> <li>• 00 0100b: x4</li> <li>• 00 1000b: x8 values are undefined when the Link is not up.</li> </ul> </li> <li>• Bit [7:6] <b>Current Link Speed</b>: indicates the negotiated Link speed of the given PCI Express Link: <ul style="list-style-type: none"> <li>• 01b: 2.5 Gbps</li> <li>• 10b: 5.0 Gbps values are undefined when the Link is not up.</li> </ul> </li> </ul>

**Table 22: Test interface signals**

## Chapter 13 Protection Logic Interface

Protocore signals are connected or not according to the type of package.

For source and full packages, as well as eval packages when the board used is not a PLDA board:

- prot2\_in is tied to 0
- prot0\_out is not connected
- prot2\_out is not connected

For eval packages when the board used is a PLDA board, pins are assigned according to the board schematics. See the *Reference Manual* for your PLDA board for more information.

## Chapter 14 Bridge Configuration Space

The Bridge Configuration Space consists of:

- the Bridge Internal Registers
- the PCIe Configuration Space, accessible through the PCIe Configuration interface
- the External Registers, accessible through the AXI4-Lite Master interface

Byte Address	R/W	Description
<b>Bridge Internal Registers (4 KBytes)</b>		
0x0000 - 0x017F	RO/RW	Control and Status Registers (see <a href="#">Section 14.1</a> )
0x0180 - 0x01FF	RO/RW	Interrupt and Event Registers (see <a href="#">Section 14.2</a> )
0x0200 - 0x02FF	RO/RW	Routing, Arbitration and Priority Rules (see <a href="#">Section 14.3</a> )
0x0300 - 0x03FF	RO/RW	Interface Optional Features Definitions (see <a href="#">Section 14.4</a> )
0x0400 - 0x05FF	RO/RW	DMA Engines Registers (see <a href="#">Section 14.5</a> )
0x0600 - 0x0BFF	RO/RW	Address Translation Registers (see <a href="#">Section 14.6</a> )
0x0C00 - 0x0FFF	RO	Reserved
<b>PCIe Configuration Space Registers (4 KBytes)</b> (see <a href="#">Section 14.7</a> and <a href="#">Appendix A: Register Content of the PCIe Config Space</a> )		
0x1000 - 0x1100	RO/RW	PCI Configuration Space
0x1100 - 0x1FFF	RO/RW	PCIe Extended Configuration Space
<b>Bridge External Registers (8 KBytes)</b> (see <a href="#">Section 14.8</a> )		
0x2000 - 0x3FFF	RO/RW	User-defined Registers

**Table 23: Bridge Configuration Space**

## 14.1 Control and Status Registers

Name	Byte address	R/W	Description
BRIDGE_VER	0x0000 - 0x0003	RO	<b>Bridge IP Version &amp; Revision:</b> <ul style="list-style-type: none"> <li>• Bit [11:0]: VERSION: provides the Bridge IP Core version; for example, 12'h123 indicates version 1.2.3 of the Core.</li> <li>• Bit[23:12]: PRODUCT_ID: provides the Bridge IP Product ID, equal to 12'h511</li> <li>• Bit[27:24]: DMA_NUM: indicates the number of DMA Engines implemented in the Core. Supported values are between 4'h0 and 4'h8.</li> <li>• Bit[31:28]: reserved</li> </ul>
BRIDGE_BUS	0x0004 - 0x0007	RO	<b>Bridge Internal Bus:</b> <ul style="list-style-type: none"> <li>• Bit [11:0]: VERSION: provides the Bridge Internal Bus version; for example, 12'h123 indicates version 1.2.3 of the Bus</li> <li>• Bit [15:12]: DATAPATH: indicates the Bridge Internal Bus Data Path width: <ul style="list-style-type: none"> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the Bridge Internal Bus</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the Bridge Internal Bus</li> </ul> <p>For the RD_OUTREQ_N and WR_OUTREQ_N fields supported values are:</p> <ul style="list-style-type: none"> <li>• 8: 256 outstanding requests</li> </ul> <p>For the MAXPAYLOAD and MAXRREQSIZE fields supported values are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
BRIDGE_IMPL_IF	0x0008 - 0x000B	RO	<b>Bridge Implemented Interface:</b> Describes the Bridge implemented interface: <ul style="list-style-type: none"> <li>• Bit [0]: PCIe Interface is implemented</li> <li>• Bit [1]: PCIe Interface Window 0 is implemented</li> <li>• Bit [2]: PCIe Interface Window 1 is implemented</li> <li>• Bit [3]: PCIe Configuration Interface is implemented</li> <li>• Bit [4]: AXI4-Lite Master Interface is implemented</li> <li>• Bit [5]: AXI4-Lite Slave Interface is implemented</li> <li>• Bit [6]: AXI4 Master Descriptor Interface is implemented</li> <li>• Bit [7]: reserved</li> <li>• Bit [8,10,12,14]: AXI4 Master 0,1,2,3 Interface is implemented</li> <li>• Bit [9,11,13,15]: AXI4 Slave 0,1,2,3 Interface is implemented</li> <li>• Bit [16,18,20,22]: AXI4 Stream Out 0,1,2,3 Interface is implemented</li> <li>• Bit [17,19,21,23]: AXI4 Stream In 0,1,2,3 Interface is implemented</li> <li>• Bit [31:24]: reserved</li> </ul> <b>Note:</b> More detailed information on PCIe Interface is provided at addresses 0x0010 - 0x001B, and more detailed information on AXI Interfaces is provided at addresses 0x0020 - 0x006F.
--	0x000C - 0x000F	RO	reserved
PCIE_IF_CONF	0x0010 - 0x0013	RO	<b>PCIe Interface Configuration:</b> <ul style="list-style-type: none"> <li>• Bit [11:0]: VERSION: provides the PCI Express Controller Core version; for example, 12'h123 indicates version 1.2.3 of the Core</li> <li>• Bit [15:12]: IF_ID: provides the ID used to target this interface: 0. This ID is used to specify the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.6</a>)</li> <li>• Bit [19:16]: B2P_MRD_OUTREQ_N: number of outstanding read requests the Bridge can issue to the PCIe domain</li> <li>• Bit [23:20]: P2B_MRD_OUTREQ_N: number of outstanding read requests from the PCIe domain the Bridge can handle simultaneously</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the PCIe Interface</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the PCIe Interface</li> </ul> Supported values for B2P_MRD_OUTREQ_N and P2B_MRD_OUTREQ_N are: <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> <li>• 1: 2 outstanding requests</li> <li>• ...</li> <li>• 7: 128 outstanding requests</li> </ul> Supported values for MAXPAYLOAD and MAXRREQSIZE are: <ul style="list-style-type: none"> <li>• 0: 128 Bytes</li> <li>• 1: 256 Bytes</li> <li>• 2: 512 Bytes</li> <li>• 3: 1024 Bytes</li> <li>• 4: 2048 Bytes</li> <li>• 5: 4096 Bytes</li> </ul>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
PCIE_BASIC_CONF	0x0014 - 0x0017	RO	<p><b>PCIe IP Basic Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [7:0]: LINK_WIDTH: advertises the supported link width: <ul style="list-style-type: none"> <li>• Bit 0: x1 configuration is supported. (This bit is always asserted.)</li> <li>• Bit 1: x2 configuration is supported</li> <li>• Bit 2: x4 configuration is supported</li> <li>• Bit 3: x8 configuration is supported</li> <li>• Bit 4: x16 configuration is supported</li> <li>• Bit 5: x32 configuration is supported</li> <li>• Bit 7-6: reserved</li> </ul> </li> <li>• <b>Note:</b> Several bits of this field can be asserted at the same time.</li> <li>• Bit [15:8]: LINK_SPEED: advertises the supported link speed: <ul style="list-style-type: none"> <li>• Bit 0: 2.5 Gbps link speed is supported</li> <li>• Bit 1: 5.0 Gbps link speed is supported</li> <li>• Bit 2: 8.0 Gbps link speed is supported</li> <li>• Bits 7-3: reserved</li> </ul> </li> <li>• <b>Note:</b> Several bits of this field can be asserted at the same time.</li> <li>• Bit [19:16]: FUNC_NUM: advertises the number of functions implemented in the Core. Supported values are between 4'h1 and 4'h8. Other values are reserved.</li> <li>• Bit [23:20]: VC_NUM: advertises the number of Virtual Channels implemented in the Core. The only supported value is 4'h; other values are reserved.</li> <li>• Bit [27:24]: COMPL: advertises the Core compliance to PCI Express 3.0 specification. The only supported value is 4'h3; other values are reserved.</li> <li>• Bit [31:28]: TYPE: advertises the PCI Express Core type. Supported values are: <ul style="list-style-type: none"> <li>• 4'h0: Native Endpoint</li> </ul> Other values are reserved. </li> </ul>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
PCIE_BASIC_STATUS	0x0018 - 0x001B	RO	<p><b>PCIe IP Basic Status:</b></p> <ul style="list-style-type: none"> <li>• Bit [7:0]: NEG_LINK_WIDTH: reports the negotiated link width of the PCIe link. Supported values are: <ul style="list-style-type: none"> <li>• 01: x1 link width</li> <li>• 02: x2 link width</li> <li>• 04: x4 link width</li> <li>• 08: x8 link width</li> <li>• 10: x16 link width</li> </ul> </li> <li>• Bit [11:8]: NEG_LINK_SPEED: reports the negotiated link speed of the PCIe link. Supported values are: <ul style="list-style-type: none"> <li>• 1: 2.5 Gbps link speed</li> <li>• 2: 5.0 Gbps link speed</li> <li>• 3: 8.0 Gbps link speed</li> </ul> </li> <li>• Bit [23:12]: reserved</li> <li>• Bit [27:24]: NEG_MAXPAYLOAD: reports the negotiated maximum Payload of the PCIe link.</li> <li>• Bit [31:28]: NEG_MAXRRQSIZE: reports the negotiated maximum Read Request Size of the PCIe link</li> </ul> <p>The supported values for the NEG_MAXPAYLOAD and NEG_MAXRRQSIZE fields are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul>
--	0x001C - 0x001F	RO	reserved
AXI_MSTL_CONF	0x0020 - 0x0023	RO	<p><b>AXI4-Lite Master Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 0: AXI4-Lite Master</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 2. This ID is used to specify the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.6</a>).</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 1: CDC is implemented</li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 2: 32-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum payload for the AXI Interface</li> <li>• Bit [31:28]: MAXRRQSIZE: provides the maximum Read Request Size for the AXI Interface</li> </ul> <p>Supported values for RD_OUTREQ_N and WR_OUTREQ_N are:</p> <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> </ul> <p>Supported values for MAXPAYLOAD and MAXRRQSIZE are:</p> <ul style="list-style-type: none"> <li>• B: 4 bytes</li> </ul>

Table 24: Control and Status Registers



Name	Byte address	R/W	Description
AXI_SLVL_CONF	0x0024 - 0x0027	RO	<b>AXI4-Lite Slave Interface Configuration:</b> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 1: AXI4-Lite Slave</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 2. This ID is used to specify the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 1: CDC is implemented</li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 2: 32-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the AXI Interface</li> </ul> <p>Supported values for RD_OUTREQ_N and WR_OUTREQ_N are:</p> <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> </ul> <p>Supported values for MAXPAYLOAD and MAXRREQSIZE are:</p> <ul style="list-style-type: none"> <li>• B: 4 bytes</li> </ul>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_MSTD_CONF	0x0028 - 0x002B	RO	<p><b>AXI4 Master Descriptor Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 2: AXI4 Master</li> <li>• 6: AXI3 Master</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 3. This ID is used to specify the SRC/DEST_ID, SG/SG2_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the AXI Interface</li> </ul> <p>Supported values for RD_OUTREQ_N and WR_OUTREQ_N are:</p> <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> </ul> <p>Supported values for MAXPAYLOAD and MAXRREQSIZE are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> When the AXI Interface type is AXI3 Master, MAXPAYLOAD and MAXRREQSIZE are equal to 128 bytes. When the AXI Interface type is AXI4 Master, MAXPAYLOAD is equal to the Bridge Internal Bus MAXPAYLOAD (see the <b>BRIDGE_BUS</b> register)</p>
--	0x002C - 0x002F	RO	reserved

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_MST0_CONF	0x0030 - 0x0033	RO	<p><b>AXI4 Master 0 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 2: AXI4 Master</li> <li>• 6: AXI3 Master</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 4. This ID is used to specify the SRC/DEST_ID, SG/SG2_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the AXI Interface</li> </ul> <p>Supported values for RD_OUTREQ_N and WR_OUTREQ_N are:</p> <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> <li>• 1: 2 outstanding requests</li> <li>• ...</li> <li>• 4: 16 outstanding requests</li> </ul> <p>Supported values for MAXPAYLOAD and MAXRREQSIZE are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> When the AXI Interface type is AXI3 Master, MAXPAYLOAD and MAXRREQSIZE are equal to 128 bytes. When the AXI Interface type is AXI4 Master, MAXPAYLOAD is equal to the Bridge Internal Bus MAXPAYLOAD, and MAXRREQSIZE is equal to the Bridge Internal Bus MAXRREQSIZE (see the <b>BRIDGE_BUS</b> register).</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_SLV0_CONF	0x0034 - 0x0037	RO	<p><b>AXI4 Slave 0 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 3: AXI4 Slave</li> <li>• 7: AXI3 Slave</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 4. This ID is used to specify the SRC/DEST_ID, SG/SG2_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the AXI Interface</li> <li>• Supported values for RD_OUTREQ_N and WR_OUTREQ_N are: <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> <li>• 1: 2 outstanding requests</li> <li>• ...</li> <li>• 4: 16 outstanding requests</li> </ul> </li> </ul> <p>Supported values for MAXPAYLOAD and MAXRREQSIZE are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> When the AXI Interface type is AXI3 Master, MAXPAYLOAD and MAXRREQSIZE are equal to 128 bytes. When the AXI Interface type is AXI4 Master, MAXPAYLOAD is equal to 4096 bytes.</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_MST1_CONF	0x0038- 0x003B	RO	<p><b>AXI4 Master 1 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 2: AXI4 Master</li> <li>• 6: AXI3 Master</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 5. This ID is used to specify the SRC/DEST_ID, SG/SG2_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the AXI Interface</li> </ul> <p>Supported values for RD_OUTREQ_N and WR_OUTREQ_N are:</p> <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> <li>• 1: 2 outstanding requests</li> <li>• ...</li> <li>• 4: 16 outstanding requests</li> </ul> <p>Supported values for MAXPAYLOAD and MAXRREQSIZE are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> When the AXI Interface type is AXI3 Master, MAXPAYLOAD and MAXRREQSIZE are equal to 128 bytes. When the AXI Interface type is AXI4 Master, MAXPAYLOAD is equal to the Bridge Internal Bus MAXPAYLOAD, and MAXRREQSIZE is equal to the Bridge Internal Bus MAXRREQSIZE (see the <b>BRIDGE_BUS</b> register).</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_SLV1_CONF	0x003C - 0x003F	RO	<p><b>AXI4 Slave 1 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 3: AXI4 Slave</li> <li>• 7: AXI3 Slave</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 5. This ID is used to specify the SRC/DEST_ID, SG/SG2_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the AXI Interface</li> </ul> <p>Supported values for RD_OUTREQ_N and WR_OUTREQ_N are:</p> <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> <li>• 1: 2 outstanding requests</li> <li>• ...</li> <li>• 4: 16 outstanding requests</li> </ul> <p>Supported values for MAXPAYLOAD and MAXRREQSIZE are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> When the AXI Interface type is AXI3 Master, MAXPAYLOAD and MAXRREQSIZE are equal to 128 bytes. When the AXI Interface type is AXI4 Master, MAXPAYLOAD is equal to 4096 bytes.</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_MST2_CONF	0x0040 - 0x0043	RO	<p><b>AXI4 Master 2 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 2: AXI4 Master</li> <li>• 6: AXI3 Master</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 6. This ID is used to specify the SRC/DEST_ID, SG/SG2_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the AXI Interface</li> </ul> <p>Supported values for RD_OUTREQ_N and WR_OUTREQ_N are:</p> <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> <li>• 1: 2 outstanding requests</li> <li>• ...</li> <li>• 4: 16 outstanding requests</li> </ul> <p>Supported values for MAXPAYLOAD and MAXRREQSIZE are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> When the AXI Interface type is AXI3 Master, MAXPAYLOAD and MAXRREQSIZE are equal to 128 bytes. When the AXI Interface type is AXI4 Master, MAXPAYLOAD is equal to the Bridge Internal Bus MAXPAYLOAD, and MAXRREQSIZE is equal to the Bridge Internal Bus MAXRREQSIZE (see the <b>BRIDGE_BUS</b> register).</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_SLV2_CONF	0x0044 - 0x0047	RO	<p><b>AXI4 Slave 2 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 3: AXI4 Slave</li> <li>• 7: AXI3 Slave</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 6. This ID is used to specify the SRC/DEST_ID, SG/SG2_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the AXI Interface</li> </ul> <p>Supported values for RD_OUTREQ_N and WR_OUTREQ_N are:</p> <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> <li>• 1: 2 outstanding requests</li> <li>• ...</li> <li>• 4: 16 outstanding requests</li> </ul> <p>Supported values for MAXPAYLOAD and MAXRREQSIZE are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> When the AXI Interface type is AXI3 Master, MAXPAYLOAD and MAXRREQSIZE are equal to 128 bytes. When the AXI Interface type is AXI4 Master, MAXPAYLOAD is equal to 4096 bytes.</p>

Table 24: Control and Status Registers



Name	Byte address	R/W	Description
AXI_MST3_CONF	0x0048- 0x004B	RO	<p><b>AXI4 Master 3 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 2: AXI4 Master</li> <li>• 6: AXI3 Master</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 7. This ID is used to specify the SRC/DEST_ID, SG/SG2_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the AXI Interface</li> </ul> <p>Supported values for RD_OUTREQ_N and WR_OUTREQ_N are:</p> <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> <li>• 1: 2 outstanding requests</li> <li>• ...</li> <li>• 4: 16 outstanding requests</li> </ul> <p>Supported values for MAXPAYLOAD and MAXRREQSIZE are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> When the AXI Interface type is AXI3 Master, MAXPAYLOAD and MAXRREQSIZE are equal to 128 bytes. When the AXI Interface type is AXI4 Master, MAXPAYLOAD is equal to the Bridge Internal Bus MAXPAYLOAD, and MAXRREQSIZE is equal to the Bridge Internal Bus MAXRREQSIZE (see the <b>BRIDGE_BUS</b> register).</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_SLV3_CONF	0x004C - 0x004F	RO	<p><b>AXI4 Slave 3 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 3: AXI4 Slave</li> <li>• 7: AXI3 Slave</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 7. This ID is used to specify the SRC/DEST_ID, SG/SG2_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests</li> <li>• Bit [23:20]: WR_OUTREQ_N: number of outstanding write requests</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: MAXRREQSIZE: provides the maximum Read Request Size of the AXI Interface</li> </ul> <p>Supported values for RD_OUTREQ_N and WR_OUTREQ_N are:</p> <ul style="list-style-type: none"> <li>• 0: 1 outstanding request</li> <li>• 1: 2 outstanding requests</li> <li>• ...</li> <li>• 4: 16 outstanding requests</li> </ul> <p>Supported values for MAXPAYLOAD and MAXRREQSIZE are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> When the AXI Interface type is AXI3 Master, MAXPAYLOAD and MAXRREQSIZE are equal to 128 bytes. When the AXI Interface type is AXI4 Master, MAXPAYLOAD is equal to 4096 bytes.</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_STRO0_CONF	0x0050 - 0x0053	RO	<p><b>AXI4 Stream Out 0 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 4: AXI4 Stream Out</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 8. This ID is used to specify the SRC/DEST_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [23:16]: reserved</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: reserved</li> </ul> <p>Supported values for MAXPAYLOAD are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> MAXPAYLOAD is equal to the Bridge Internal Bus MAXPAYLOAD (see the <b>BRIDGE_BUS</b> register).</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_STRI0_CONF	0x0054 - 0x0057	RO	<p><b>AXI4 Stream In 0 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 5: AXI4 Stream In</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 8. This ID is used to specify the SRC/DEST_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests from the Bridge (from DMA engines or through Address Translation) that the AXI4 Stream In Interface can handle simultaneously</li> <li>• Bit [23:20]: TIDDEST_N: number of TID/TDEST combinations that the AXI4 Stream In Interface can handle simultaneously</li> <li>• Bit [27:24]: LENGTH_SUPPORT: provides the packet length supported by the AXI Stream In Interface: <ul style="list-style-type: none"> <li>• 0: all Stream In length are supported</li> <li>• 1: length is a multiple of 4 bytes</li> <li>• 2: length is a multiple of 8 bytes</li> <li>• 3: length is a multiple of the Bridge Internal Bus Data Path width (see the <b>BRIDGE_BUS</b> register).</li> </ul> </li> <li>• Bit [31:28]: RREQSIZE_SUPPORT: provides the read request size from the Bridge (from DMA engines or through Address Translation) supported by the AXI Stream In Interface: <ul style="list-style-type: none"> <li>• 0: all Stream In read request size are supported</li> <li>• 1: read request size is a multiple of 4 bytes</li> <li>• 2: read request size is a multiple of 8 bytes</li> <li>• 3: read request size is a multiple of the Bridge Internal Bus Data Path width (see the <b>BRIDGE_BUS</b> register).</li> </ul> </li> </ul> <p><b>Note:</b> The higher the LENGTH_SUPPORT and RREQSIZE_SUPPORT values are, the better frequencies the Core will achieve.</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_STRO1_CONF	0x0058 - 0x005B	RO	<p><b>AXI4 Stream Out 1 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 4: AXI4 Stream Out</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 9. This ID is used to specify the SRC/DEST_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [23:16]: reserved</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface <ul style="list-style-type: none"> <li>• Bit [31:28]: reserved</li> </ul> </li> </ul> <p>Supported values for MAXPAYLOAD are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> MAXPAYLOAD is equal to the Bridge Internal Bus MAXPAYLOAD (see the <b>BRIDGE_BUS</b> register).</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_STRI1_CONF	0x005C - 0x005F	RO	<p><b>AXI4 Stream In 1 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 5: AXI4 Stream In</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: 9. This ID is used to specify the SRC/DEST_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests from the Bridge (from DMA engines or through Address Translation) that the AXI4 Stream In Interface can handle simultaneously</li> <li>• Bit [23:20]: TIDDEST_N: number of TID/TDEST combinations that the AXI4 Stream In Interface can handle simultaneously</li> <li>• Bit [27:24]: LENGTH_SUPPORT: provides the packet length supported by the AXI Stream In Interface: <ul style="list-style-type: none"> <li>• 0: all Stream In length are supported</li> <li>• 1: length is a multiple of 4 bytes</li> <li>• 2: length is a multiple of 8 bytes</li> <li>• 3: length is a multiple of the Bridge Internal Bus Data Path width (see the <b>BRIDGE_BUS</b> register).</li> </ul> </li> <li>• Bit [31:28]: RREQSIZE_SUPPORT: provides the read request size from the Bridge (from DMA engines or through Address Translation) supported by the AXI Stream In Interface: <ul style="list-style-type: none"> <li>• 0: all Stream In read request size are supported</li> <li>• 1: read request size is a multiple of 4 bytes</li> <li>• 2: read request size is a multiple of 8 bytes</li> <li>• 3: read request size is a multiple of the Bridge Internal Bus read Data Path width (see the <b>BRIDGE_BUS</b> register).</li> </ul> </li> </ul> <p><b>Note:</b> The higher the LENGTH_SUPPORT and RREQSIZE_SUPPORT values are, the better frequencies the Core will achieve.</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_STRO2_CONF	0x0060 - 0x0063	RO	<p><b>AXI4 Stream Out 2 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 4: AXI4 Stream Out</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: A. This ID is used to specify the SRC/DEST_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [23:16]: reserved</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: reserved</li> </ul> <p>Supported values for MAXPAYLOAD are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> MAXPAYLOAD is equal to the Bridge Internal Bus MAXPAYLOAD (see the <b>BRIDGE_BUS</b> register).</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_STRI2_CONF	0x0064 - 0x0067	RO	<p><b>AXI4 Stream In 2 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 5: AXI4 Stream In</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: A. This ID is used to specify the SRC/DEST_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests from the Bridge (from DMA engines or through Address Translation) that the AXI4 Stream In Interface can handle simultaneously</li> <li>• Bit [23:20]: TIDDEST_N: number of TID/TDEST combinations that the AXI4 Stream In Interface can handle simultaneously</li> <li>• Bit [27:24]: LENGTH_SUPPORT: provides the packet length supported by the AXI Stream In Interface: <ul style="list-style-type: none"> <li>• 0: all Stream In length are supported</li> <li>• 1: length is a multiple of 4 bytes</li> <li>• 2: length is a multiple of 8 bytes</li> <li>• 3: length is a multiple of the Bridge Internal Bus Data Path width (see the <b>BRIDGE_BUS</b> register).</li> </ul> </li> <li>• Bit [31:28]: RREQSIZE_SUPPORT: provides the read request size from the Bridge (from DMA engines or through Address Translation) supported by the AXI Stream In Interface: <ul style="list-style-type: none"> <li>• 0: all Stream In read request size are supported</li> <li>• 1: read request size is a multiple of 4 bytes</li> <li>• 2: read request size is a multiple of 8 bytes</li> <li>• 3: read request size is a multiple of the Bridge Internal Bus Data Path width (see the <b>BRIDGE_BUS</b> register).</li> </ul> </li> </ul> <p><b>Note:</b> The higher the LENGTH_SUPPORT and RREQSIZE_SUPPORT values are, the better frequencies the Core will achieve.</p>

Table 24: Control and Status Registers



Name	Byte address	R/W	Description
AXI_STRO3_CONF	0x0068 - 0x006B	RO	<p><b>AXI4 Stream Out 3 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 4: AXI4 Stream Out</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: B. This ID is used to specify the SRC/DEST_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [23:16]: reserved</li> <li>• Bit [27:24]: MAXPAYLOAD: provides the maximum Payload of the AXI Interface</li> <li>• Bit [31:28]: reserved</li> </ul> <p>Supported values for MAXPAYLOAD are:</p> <ul style="list-style-type: none"> <li>• 0: 128 bytes</li> <li>• 1: 256 bytes</li> <li>• 2: 512 bytes</li> <li>• 3: 1024 bytes</li> <li>• 4: 2048 bytes</li> <li>• 5: 4096 bytes</li> </ul> <p><b>Note:</b> MAXPAYLOAD is equal to the Bridge Internal Bus MAXPAYLOAD (see the <b>BRIDGE_BUS</b> register).</p>

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
AXI_STRI3_CONF	0x006C - 0x006F	RO	<p><b>AXI4 Stream In 3 Interface Configuration:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: IF_TYPE: specifies the AXI Interface type: <ul style="list-style-type: none"> <li>• 5: AXI4 Stream In</li> </ul> </li> <li>• Bit [7:4]: IF_ID: provides the ID used to target this interface: B. This ID is used to specify the SRC/DEST_ID and the TRSL_ID fields of the DMA Engines and Address Translation registers (see <a href="#">Section 14.5</a> and <a href="#">Section 14.6</a>)</li> <li>• Bit [11:8]: CLK_DOM: sets the Clock Domain of the AXI Interface: <ul style="list-style-type: none"> <li>• 0: CDC is bypassed</li> <li>• 1-4: CDC is implemented with FIFO Depth equal to <math>2^{\text{CLK\_DOM}}</math></li> </ul> </li> <li>• Bit [15:12]: DATAPATH: provides the AXI Interface Data Path width: <ul style="list-style-type: none"> <li>• 3: 64-bits</li> <li>• 4: 128-bits</li> <li>• 5: 256-bits</li> </ul> </li> <li>• Bit [19:16]: RD_OUTREQ_N: number of outstanding read requests from the Bridge (from DMA engines or through Address Translation) that the AXI4 Stream In Interface can handle simultaneously</li> <li>• Bit [23:20]: TIDDEST_N: number of TID/TDEST combinations that the AXI4 Stream In Interface can handle simultaneously</li> <li>• Bit [27:24]: LENGTH_SUPPORT: provides the packet length supported by the AXI Stream In Interface: <ul style="list-style-type: none"> <li>• 0: all Stream In length are supported</li> <li>• 1: length is a multiple of 4 bytes</li> <li>• 2: length is a multiple of 8 bytes</li> <li>• 3: length is a multiple of the Bridge Internal Bus Data Path width (see the <b>BRIDGE_BUS</b> register).</li> </ul> </li> <li>• Bit [31:28]: RREQSIZE_SUPPORT: provides the read request size from the Bridge (from DMA engines or through Address Translation) supported by the AXI Stream In Interface: <ul style="list-style-type: none"> <li>• 0: all Stream In read request size are supported</li> <li>• 1: read request size is a multiple of 4 bytes</li> <li>• 2: read request size is a multiple of 8 bytes</li> <li>• 3: read request size is a multiple of the Bridge Internal Bus Data Path width (see the <b>BRIDGE_BUS</b> register).</li> </ul> </li> </ul> <p><b>Note:</b> The higher the LENGTH_SUPPORT and RREQSIZE_SUPPORT values are, the better frequencies the Core will achieve.</p>
--	0x0070 - 0x0173	RO	reserved

Table 24: Control and Status Registers

Name	Byte address	R/W	Description
PM_CONF	0x0174 - 0x017F	RO	<p><b>PCI Power Management Data Register:</b> When this register is not hardwired by Core Constants, it is read/write and the local processor must initialize it at power-up.</p> <p>PCI Power Management Data Register provides the scaling factor and state dependant data related to the power state selected by the Data_Select field of Power Management Control and Status register:</p> <ul style="list-style-type: none"> <li>• Bit [15:0]: reserved</li> <li>• Bit [17:16]: data scale value returned when Data_Select = 0</li> <li>• ...</li> <li>• Bit [31:30]: data scale value returned when Data_Select = 7</li> <li>• Bit [39:32]: data register value returned when Data_Select = 0</li> <li>• ...</li> <li>• Bit [95:87]: data register value returned when Data_Select = 7</li> </ul> <p>These values indicate power consumption of the component and can only be implemented if all three bits of AUX_power (part of the Power Management Capabilities structure) are set to 0. It includes:</p> <ul style="list-style-type: none"> <li>• Data Register: value associated with the power consumed by the component.</li> <li>• Data Scale: the scale used to find the power consumed by a particular component and can include the following values: <ul style="list-style-type: none"> <li>• 00: unknown</li> <li>• 01: 0.1 x</li> <li>• 10: 0.01 x</li> <li>• 11: 0.001 x</li> </ul> </li> </ul> <p>The maximum power permitted to the component in the power state selected by the Data_Select field by the component is equal to the Data Register value multiplied.</p> <p>See the <i>PCI Power Management Specification, v1.2</i> for a full description of this register.</p>

Table 24: Control and Status Registers

## 14.2 Interrupt and Event Registers

These registers are used to enable, disable, monitor and clear interrupt sources

Name	Byte Address	R/W	Description
IMASK_LOCAL	0x0180 - 0x0183	RW/RO	<b>Local Processor Interrupt Mask:</b> When this register is not hardwired by Core Constants (see <a href="#">Section 5.1.1</a> ), it is read/write and its default value after reset is 32'h0. Setting a bit enables the associated interrupt source and clearing a bit masks the interrupt source. See ISTATUS_LOCAL for details of this register's bits.

**Table 25: Interrupt and Event Registers**

Name	Byte Address	R/W	Description
ISTATUS_LOCAL	0x0184 - 0x0187	RW1C	<p><b>Local Processor Interrupt Status:</b></p> <p>This is a read/write/clear register; the register's bits are automatically set when the corresponding interrupt source is activated. Each source is independent and thus multiple sources may be active simultaneously. The local processor can monitor and clear status bits: writing 1 clears a bit, writing 0 has no effect.</p> <p>If one or more ISTATUS_LOCAL interrupt sources are active and not masked by IMASK_LOCAL, the QuickPCIe Expert issues an interrupt towards the Local Processor (on the AXI domain). The interrupt is sent through the QPCIE_INTERRUPT_OUT [31:0] output port.</p> <p>Note that an Address Translation event or DMA transfer can be reported to ISTATUS_LOCAL, ISTATUS_HOST, both, or neither, according to the Address Translation Table and the DMA Engine IRQ_ID field configuration. See <a href="#">Chapter 10</a> for more information.</p> <p>This register is composed of the following bits:</p> <ul style="list-style-type: none"> <li>• Bit [7:0]: <b>DMA_END</b>: reports that a DMA transfer is ended. Bit number <i>i</i> corresponds to DMA Engine number <i>i</i>.</li> <li>• Bit [15:8]: <b>DMA_ERROR</b>: reports that an error occurred during a DMA transfer. Bit number <i>i</i> corresponds to DMA Engine number <i>i</i>.</li> <li>• Bit [19:16]: <b>A_ATR_EVT</b>: reports AXI Address Translation events: <ul style="list-style-type: none"> <li>• Bit 3: AXI Doorbell: asserted when an AXI request has successfully targeted an Address Translation Table</li> <li>• Bit 2: AXI Discard Error: asserted to signal a completion timeout on an AXI read request</li> <li>• Bit 1: AXI Fetch Error: asserted to indicate that an error occurred on an AXI read request</li> <li>• Bit 0: AXI Post Error: asserted to indicate that an error occurred on an AXI write request</li> </ul> </li> <li>• Bit [23:20]: <b>P_ATR_EVT</b>: reports PCIe Address Translation events: <ul style="list-style-type: none"> <li>• Bit [3]: PCIe Doorbell: asserted when a PCIe request has successfully targeted an Address Translation Table</li> <li>• Bit [2]: PCIe Discard Error: asserted to signal a completion timeout on a PCIe read request</li> <li>• Bit [1]: PCIe Fetch Error: asserted to indicate that an error occurred on a PCIe read request</li> <li>• Bit [0]: PCIe Post Error: asserted to indicate that an error occurred on a PCIe write request</li> </ul> </li> <li>• Bit [31:24]: <b>PM_MSI_INT</b>: reports Power Management, MSI and Interrupts events to the Local Processor: <ul style="list-style-type: none"> <li>• Bit [7]: reserved</li> <li>• Bit [6]: Legacy power management state change</li> <li>• Bit [5:0]: reserved</li> </ul> </li> </ul> <p>When one of these interrupt's sources is activated, an interrupt is issued to the local processor.</p>

Table 25: Interrupt and Event Registers

Name	Byte Address	R/W	Description
IMASK_HOST	0x0188 - 0x018B	RW/RO	<p><b>Host Processor Interrupt Mask:</b> When this register is not hardwired by Core Constants (see <a href="#">Chapter 5</a>), it is read/write and its default value after reset is 32'h0. Setting a bit enables the associated interrupt source and clearing a bit masks the interrupt source.</p> <p>See ISTATUS_HOST for details of this register's bits.</p>
ISTATUS_HOST	0x018C - 0x018F	RW1C/RO	<p><b>Host Processor Interrupt Status:</b></p> <p>This is a read/write/clear register; the register's bits are automatically set when the corresponding interrupt source is activated. Each source is independent and thus multiple sources may be active simultaneously. The host processor can monitor and clear status bits: writing 1 clears a bit, writing 0 has no effect.</p> <p>If one or more ISTATUS_HOST interrupt sources are active and not masked by IMASK_HOST, the Bridge IP Core issues an interrupt towards the Host Processor (on the PCIe domain). The interrupt is sent using Message Signalled Interrupt if the PCI host processor has enabled MSI, otherwise INT messages are used.</p> <p>Note that an Address Translation event or DMA transfer can be reported to ISTATUS_LOCAL, ISTATUS_HOST, both, or neither, according to the Address Translation Table and the DMA Engine IRQ_ID field configuration. See <a href="#">Chapter 10</a> for more information.</p> <p>This register is composed of the following bits:</p> <ul style="list-style-type: none"> <li>• Bit [7:0]: <b>DMA_END</b> (see ISTATUS_LOCAL)</li> <li>• Bit [15:8]: <b>DMA_ERROR</b> (see ISTATUS_LOCAL)</li> <li>• Bit [19:16]: <b>A_ATR_EVT</b> (see ISTATUS_LOCAL)</li> <li>• Bit [23:20]: <b>P_ATR_EVT</b> (see ISTATUS_LOCAL)</li> <li>• Bit [31:24]: <b>INT_REQUEST</b>: reports interrupt requests from the local processor to the Host Processor.</li> </ul> <p>When the PCIe is Endpoint, the local processor can drive up to 8 interrupt sources high by generating a pulse (high) on the QPCIE_INTERRUPT_IN[7:0] input port, and can drive them low by writing 1 to this bit.</p>
IMSI_ADDR	0x0190 - 0x0193	RO	<b>MSI Capture Address</b> reserved
ISTATUS_MSI	0x0194 - 0x0197	RO/RW1C	<b>MSI Message</b> reserved
ICMD_PM	0x0198 - 0x019B	RW	<p><b>Event Command:</b> enables the local processor to activate and send events to the PCIe bus:</p> <ul style="list-style-type: none"> <li>• Bit [31-5]: reserved</li> <li>• Bit [4]: reserved</li> <li>• Bit [3-1]: reserved</li> <li>• Bit [0]: Send PME: Local processor can write 1 to send PME bit command in order to generate a PME# event on the PCI Express link. This is used to ask PCI host processor to restore bridge to a fully functional legacy power state. Note that this bit is immediately cleared by the QuickPCIe Expert Core: the local processor does not need to clear it.</li> </ul> <p>Note that the PME_En bit in the Power Management Control/Status register should be set to 1b to enable PME# to be asserted (see <a href="#">Section A.1.4</a>).</p>

Table 25: Interrupt and Event Registers

Name	Byte Address	R/W	Description
ISTATUS_PM	0x019C - 0x019F	RO	<p><b>PCI Legacy Power Management State:</b> reports the PCIe power state:</p> <ul style="list-style-type: none"> <li>• Bit 1-0: This register specifies the PCI Legacy Power Management state (00=D0, 01=D1, 10=D2 or 11=D3hot or D3cold). It is used when the PCIe is Endpoint. Note that change in the power management state is reported by an interrupt in the ISTATUS_LOCAL register (bit 6 of PM_MSI_INT Field)</li> <li>• Bits 31-2: reserved</li> </ul>
--	0x01A0 - 0x01AF	RO	reserved
ISTATUS_DMA0	0x01B0 - 0x01B3	RO	These registers are copies of DMA Engine's DMA_STATUS registers (see <a href="#">Section 14.5</a> ). They are automatically cleared when Bit 0 of Control Field is set to 1b.
ISTATUS_DMA1	0x01B4 - 0x01B7	RO	
ISTATUS_DMA2	0x01B8 - 0x01BB	RO	
ISTATUS_DMA3	0x01BC - 0x01BF	RO	
ISTATUS_DMA4	0x01C0 - 0x01C3	RO	
ISTATUS_DMA5	0x01C4 - 0x01C7	RO	
ISTATUS_DMA6	0x01C8 - 0x01CB	RO	
ISTATUS_DMA7	0x01CC - 0x01CF	RO	
--	0x01D0- 0x01D3	RO	reserved
--	0x01D4- 0x01D7	RO	reserved
ISTATUS_P_ADT_WIN0	0x01D8- 0x01DB	RO	<p>These registers are read only registers. The status register's bits are automatically set when a corresponding event occurs. Each event is independent, so multiple bits may be active simultaneously.</p> <p>These registers are composed of 8 vectors of 4-bits (ADT_0 - ADT_7). The vector number reports the address translation events related to the Address Translation Table number (see <a href="#">Section 14.6</a>).</p> <p>Each vector has the same mapping as P_ATR_EVT and A_ATR_EVT Field (see ISTATUS_LOCAL).</p> <p>These registers are automatically cleared when the corresponding bits of P_ATR_EVT and A_ATR_EVT Field are written to 1b.</p>
ISTATUS_P_ADT_WIN1	0x01DC- 0x01DF	RO	
ISTATUS_A_ADT_SLV0	0x01E0 - 0x01E3	RO	
ISTATUS_A_ADT_SLV1	0x01E4 - 0x01E7	RO	
ISTATUS_A_ADT_SLV2	0x01E8 - 0x01EB	RO	
ISTATUS_A_ADT_SLV3	0x01EC - 0x01EF	RO	
--	0x01F0- 0x01FF	RO	reserved

Table 25: Interrupt and Event Registers

## 14.3 Routing, Arbitration and Priority Rules Registers

Name	Address	R/W	Description
ROUTING_RULES_R	0x0200 - 0x023F	RO	<p><b>Routing Rules:</b></p> <p>Routing Rules: defines the allowed connections between Read Requester and Read Completer modules. It consists of 16 vectors of 32 bits.</p> <p>When bit <b>j</b> of vector <b>i</b> is asserted, it means that Completer <b>i</b> is allowed to receive a read request from Requester <b>j</b>, and that Requester <b>j</b> is allowed to receive a read completion from Completer <b>i</b>.</p> <p>Its value is equal to {QPCIE_KARB15_RREQ_M[31:0] , .. ,QPCIE_KARB00_RREQ_M[31:0]} (see <a href="#">Section 5.1.2</a>).</p>
ROUTING_RULES_W	0x0240 - 0x027F	RO	<p><b>Routing Rules:</b> defines the allowed connections between Write Requester and Write Completer modules. It consists of 16 vectors of 32 bits.</p> <p>When bit <b>j</b> of vector <b>i</b> is asserted, it means that Completer <b>i</b> is allowed to receive a write request from Requester <b>j</b>, and that Requester <b>j</b> is allowed to receive a write completion from Completer <b>i</b>.</p> <p>Its value is equal to {QPCIE_KARB15_WREQ_M[31:0] , .. , QPCIE_KARB00_WREQ_M[31:0]} (see <a href="#">Section 5.1.2</a>).</p>
ARBITRATION_RULES	0x0280 - 0x02BF	RO or RW	<p><b>Arbitration Rules:</b> defines the implemented arbitration type for each Requester/Completer. It consists of 32 vectors of 16 bits each, with each vector composed of 4 sub-fields:</p> <ul style="list-style-type: none"> <li>• <b>RdReqArbType:</b> defines the arbitration type for read requests</li> <li>• <b>RdCplArbType:</b> defines the arbitration type for read completions</li> <li>• <b>WrReqArbType:</b> defines the arbitration type for write requests</li> <li>• <b>WrCplArbType:</b> defines the arbitration type for write completions</li> </ul> <p>Supported value for these sub-fields are:</p> <ul style="list-style-type: none"> <li>• 0: Fixed Arbitration</li> <li>• 1: Round Robin Arbitration</li> <li>• 2: Weighted Round Robin Arbitration</li> </ul> <p>When this register is not hardwired by Core Constants, it is read/write and its default value after reset is {(32){16'h1111}}.</p>

Table 26: Routing, Arbitration, and Priority Rules Registers



Name	Address	R/W	Description
PRIORITY_RULES	0x02C0 - 0x02FF	RO or RW	<p><b>Priority Rules:</b></p> <p>When Weighted Round Robin arbitration is implemented, the Priority Rules define which Requester/Completer should be given the highest priority.</p> <p>This register consists of 32 vectors of 16-bits each, with each vector composed of 4 sub-fields:</p> <ul style="list-style-type: none"> <li>• <b>RdReqPriority:</b> defines the priority of the read request</li> <li>• <b>RdCplPriority:</b> defines the priority of the read completion</li> <li>• <b>WrReqPriority:</b> defines the priority of the write request</li> <li>• <b>WrCplPriority:</b> defines the priority of the write completion</li> </ul> <p>The higher the Priority value attributed to a request or completion, the lower the actual priority. For example, a request with a priority value of 2 is higher priority than a request with a value of 4.</p> <p>When this register is not hardwired by Core Constants, it is read/write and its default value after reset is 512'h0.</p>

Table 26: Routing, Arbitration, and Priority Rules Registers

## 14.4 Optional Features Registers

These registers define AXI (or PCIe) optional features. When they are not hardwired by Core Constants (see [Chapter 5](#)), these registers are read/write and their default values after reset are 128'h0.

Name	Byte Address	R/W	Description
--	0x0300 - 0x37F	RO or RW	Reserved
AXI4_STRIN0_MODE	0x0380 - 0x038F	RO or RW	<p>Each AXI4 Stream Interface Module can manage up to 8 different input flows, identified by their TID/TDEST combination. You can configure the number of different input flows by setting the QPCIE_AXI_STR0-3_IDDEST_N Core Constant (see <a href="#">Chapter 5</a>).</p> <p>Each AXI4_STRINx_MODE register defines how these input flows are managed. Each register is composed of 8 vectors of 16 bits (one vector per TID/TDEST combination).</p> <ul style="list-style-type: none"> <li>• Bit [0]: FIX_TX_VALUE: When asserted, the TID/TDEST combination value is fixed and defined by FIX_TID_VALUE and FIX_TDEST_VALUE. When deasserted, the Stream In Interface will capture any TID/TDEST combination, and will keep locked on this combination as long as the input flow is not completely forwarded to its destination, allowing almost any TID/TDEST combination to be handled.</li> <li>• Bit [3:1]: reserved</li> <li>• Bit [11:4]: FIX_TID_VALUE</li> <li>• Bit [15:12]: FIX_TDEST_VALUE</li> </ul>
AXI4_STRIN1_MODE	0x0390 - 0x039F	RO or RW	
AXI4_STRIN2_MODE	0x03A0 - 0x03AF	RO or RW	
AXI4_STRIN3_MODE	0x03B0 - 0x03BF	RO or RW	
--	0x03C0 - 0x03FF	RO	Reserved

Table 27: Optional Features Registers

## 14.5 DMA Engines Registers

The DMA Engine registers enable up to 8 fully independent DMA Engines to be configured:

Byte Address	R/W	Description
0x0400 - 0x043F	RW/RO	DMA Engine 0 Configuration Space
0x0440 - 0x047F	RW/RO	DMA Engine 1 Configuration Space
0x0480 - 0x04BF	RW/RO	DMA Engine 2 Configuration Space
0x04C0 - 0x04FF	RW/RO	DMA Engine 3 Configuration Space
0x0500 - 0x053F	RW/RO	DMA Engine 4 Configuration Space
0x0540 - 0x057F	RW/RO	DMA Engine 5 Configuration Space
0x0580 - 0x05BF	RW/RO	DMA Engine 6 Configuration Space
0x05C0 - 0x05FF	RW/RO	DMA Engine 7 Configuration Space

**Table 28: DMA Engine Registers**

Each DMA Engine Configuration Space is 64 bytes long and has been mapped into two 32-bytes sections; one for status and the other for control.

This enables the DMA transfer to be configured or monitored with a single 32-bytes write burst, rather than several write requests.

The following table describes configuration space registers:

Name	Byte Offset	R/W	Description
<b>DMA_SRCPARAM</b>	0x00 - 0x03	RW/RO	<p><b>Source Parameters:</b></p> <p>DMA_SRCPARAM is used to configure the source of the DMA Transfer, the transfer priority inside the Bridge IP Core, and the transfer parameters:</p> <ul style="list-style-type: none"> <li>• Bit [3:0]: <b>SRC_ID</b>: defines the Source interface ID of the DMA Transfer: <ul style="list-style-type: none"> <li>• 4'h0: PCIe Interface</li> <li>• 4'h4 - 4'h7: AXI4-Master Interface Number 0-3</li> <li>• 4'h8 - 4'hB: AXI4-Stream Interface Number 0-3</li> </ul> </li> </ul> <p>When these register fields are not hardwired by Core Constants, they are read/write and their default value after reset is 4'h0.</p> <ul style="list-style-type: none"> <li>• Bit [7:4]: reserved</li> <li>• Bit [11:8]: <b>TRSF_PRIOR</b>: defines the Transfer Priority inside the QuickPCIe Expert. It is only relevant when Weighted Round Robin arbitration is implemented. The higher the priority value attributed, the lower the actual priority given.</li> <li>• Bit [15:12]: reserved</li> <li>• Bit [27:16]: <b>TRSF_PARAM</b>: provides the Transfer Parameters (see <a href="#">Section 7.2</a>). Its content depends on the value of SRC_ID.</li> <li>• Bit [31:28]: reserved.</li> </ul>

**Table 29: DMA Engine Configuration Space Registers**

Name	Byte Offset	R/W	Description
<b>DMA_DESTPARAM</b>	0x04 - 0x07	RW/RO	<p><b>Destination Parameters:</b></p> <p>DMA_DESTPARAM is used to configure the destination of the DMA Transfer, the transfer priority inside the Bridge IP Core, and the transfer parameters:</p> <ul style="list-style-type: none"> <li>• Bit [3:0]: <b>DEST_ID</b>: defines the Destination interface ID of the DMA Transfer: <ul style="list-style-type: none"> <li>• 4'h0: PCIe Interface</li> <li>• 4'h4 - 4'h7: AXI4-Master Interface Number 0-3</li> <li>• 4'h8 - 4'hB: AXI4-Stream Interface Number 0-3</li> </ul> </li> <li>• When these register fields are not hardwired by Core Constants, they are read/write and their default value after reset is 4'h0.</li> <li>• Bit [7:4]: reserved</li> <li>• Bit [11:8]: <b>TRSF_PRIOR</b>: defines the Transfer Priority inside the QuickPCIe Expert. It is only relevant when Weighted Round Robin arbitration is implemented. The higher the priority value attributed, the lower the actual priority given.</li> <li>• Bit [15:12]: reserved</li> <li>• Bit [27:16]: <b>TRSF_PARAM</b>: provides the Transfer Parameters (see <a href="#">Section 7.2</a>). Its content depends on the value of DEST_ID.</li> <li>• Bit [31:28]: reserved.</li> </ul>
<b>DMA_SRCADDR[31: 0]</b>	0x08 - 0x0B	RW	<p><b>Source Address [31: 0], Source Address[63:32], Destination Address [31: 0], Destination Address [63:32]:</b></p> <p>These registers provide the starting source and destination address of the DMA Transfer (if relevant). If the actual source and destination address are less than 64-bits, MSB bits are ignored.</p> <ul style="list-style-type: none"> <li>• When SG mode is enabled for the source (see DMA_CONTROL and DMA_LENGTH registers), DMA_SRCADDR provides the address of the first Source descriptor.</li> <li>• When SG mode is enabled for the destination, DMA_DESTADDR provides the address of the first Destination descriptor.</li> </ul> <p>Note that when SG Type Field is equal to 2'b11, DMA_DESTADDR is irrelevant as it is identical to DMA_SRCADDR.</p>
<b>DMA_SRCADDR[63:32]</b>	0x0C - 0x0F	RW	
<b>DMA_DESTADDR[31: 0]</b>	0x10 - 0x13	RW	
<b>DMA_DESTADDR [63:32]</b>	0x14 - 0x17	RW	
<b>DMA_LENGTH</b>	0x18 - 0x1B	RW	<p><b>DMA Length:</b> DMA_LENGTH Register provides the amount of data in bytes (up to 4GB) that should be transferred from the Source to the Destination. If Bit 1 of the SE_COND Field is cleared, this value is only indicative and will be used to compute Bit 0 of the STATUS Field of the DMA_STATUS register.</p>

Table 29: DMA Engine Configuration Space Registers

Name	Byte Offset	R/W	Description
DMA_CONTROL	0x1C - 0x1F	RW	<p><b>DMA Control:</b></p> <ul style="list-style-type: none"> <li>• Bit [3:0]: <b>CTRL</b>: provides the basic controls of the DMA: <ul style="list-style-type: none"> <li>• Bit [0]: Start/Abort</li> <li>• : When set to 1b, it launches the DMA transfer; appropriate registers should have previously been set. This bit is automatically cleared by the DMA Engine at the end of the DMA transfer. If the transfer is not ended and a user sets it to 0, the transfer is aborted (see <a href="#">Section 7.5</a>).</li> <li>• Bit [1]: Pause/Resume: optional bit to enable a DMA transfer to be paused (to temporarily give more bandwidth to a transfer with higher priority).</li> <li>• Bit [2]: reserved</li> <li>• Bit [3]: enables SG mode. When this register field is not hardwired by Core Constants, it is read/write and its default value after reset is 0b.</li> </ul> </li> <li>• Bit [7:4]: <b>SE_COND</b>: defines the Start and End conditions of the DMA: <ul style="list-style-type: none"> <li>• Bit 0: Start on SOP reception (only relevant when the source is an AXI4 Stream Interface that is on data following a TLAST assertion).</li> <li>• Bit 1: Stop if DMA_LENGTH is reached.</li> <li>• Bit 2: If the DMA destination is an AXI-Stream Interface, generate an EOP at the end of the DMA. If the source of the DMA is an AXI-Stream Interface, stop the DMA if the source of the transfer reports an EOP condition.</li> <li>• Bit 3: Abort on error condition (otherwise erroneous packet or descriptor is considered as processed, but the error is logged in source and/or destination error fields).</li> </ul> <p>Note that if bits [3:1] SE_COND Field are 3'b000, a DMA transfer can only stop on a user abort or if SG mode is enabled and an End Of Chain is received.</p> </li> <li>• Bit [11:8]: <b>IRQ</b>: defines when an interrupt should be issued: <ul style="list-style-type: none"> <li>• Bit 0: an IRQ is issued on a DMA end.</li> <li>• Bit 1: an IRQ is issued if an error occurs.</li> <li>• Bit 2: an IRQ is issued if the source of the transfer reports an EOP condition.</li> <li>• Bit 3: reserved</li> </ul> <p>Note that if IRQ is set to 0, no interrupt will be issued.</p> </li> <li>• Bit [13:12]: <b>IRQ_ID</b>: defines on which interface DMA transfer events should be reported: <ul style="list-style-type: none"> <li>• Bit 1: interrupt is issued to the Host Processor (on PCIe domain).</li> <li>• Bit 0: interrupt is issued to the Local Processor (on AXI domain).</li> </ul> <p>Note that both these bits cannot be set to '1' at the same time; to generate an interruption on both sides, you must enable interrupts on the AXI side, and use <code>QPCIE_INTERRUPT_IN</code> to generate interrupts on the PCIe side (see <a href="#">Chapter 10</a>).</p> <p>If a DMA transfer event occurs, it is reported to ISTATUS_HOST and/or ISTATUS_LOCAL registers (see the ISTATUS register), depending on the IRQ_ID content.</p> </li> <li>• Bit [22:14]: reserved</li> <li>• Bit [23]: <b>DESC_UPDT</b>: set to 1 by the application to indicate to the DMA Engine that a Descriptor has been updated. It is only relevant when SG mode is enabled and the DESC_RDY field was set to 0 (see DESC_RDY). This bit is automatically cleared by the DMA Engine.</li> </ul>

Table 29: DMA Engine Configuration Space Registers

Name	Byte Offset	R/W	Description
<b>DMA_CONTROL</b> (continued)	0x1C - 0x1F	RW	<ul style="list-style-type: none"> <li>• Bit [25:24]: <b>SG_TYPE</b>: defines the Scatter-Gather type for the DMA (only relevant if bit 3 of CTRL Field is set). <ul style="list-style-type: none"> <li>• 2'b00: independent SG for both Source and Destination</li> <li>• 2'b01: Source address is set according to Descriptor, Destination address is incremented</li> <li>• 2'b10: Destination address is set according to Descriptor, Source address is incremented</li> <li>• 2'b11: Source and Destination addresses are set according to Descriptor</li> </ul> </li> </ul> <p>See <a href="#">Section 14.6</a> for illustrations of the different SG types.</p> <p>When this register field is not hardwired by Core Constants, it is read/write and its default value after reset is 2'b0.</p> <ul style="list-style-type: none"> <li>• Bit [28:26]: <b>SG_ID</b>: define on which interface the descriptors should be read: <ul style="list-style-type: none"> <li>• 3'h0: PCIe Interface</li> <li>• 3'h3: AXI4-Master Descriptor Interface</li> <li>• 3'h4 - 4'h7: AXI4-Master Interface Number 0-3</li> </ul> </li> <li>• Bit [31:29]: <b>SG2_ID</b>: as for SG_ID</li> </ul> <p>Note that if SG_TYPE is set to 2'b00, SG_ID is connected to the Source and SG2_ID is connected to the Destination. Otherwise, SG_ID is connected to the Source and/or Destination, and SG2_ID is irrelevant.</p> <p>When the SG_ID and SG2_ID registers are not hardwired by Core Constants, they are read/write and their default value after reset is 'b0.</p>

Table 29: DMA Engine Configuration Space Registers

Name	Byte Offset	R/W	Description
<b>DMA_STATUS</b>	0x20 - 0x23	RO	<p><b>DMA_STATUS:</b></p> <ul style="list-style-type: none"> <li>• Bit [7:0]: <b>STATUS:</b> <ul style="list-style-type: none"> <li>• Bit 0: DMA Complete with DMA_LENGTH reached</li> <li>• Bit 1: DMA Complete with an EOP condition reported by the source of the transfer</li> <li>• Bit 2: DMA Complete with EOC received on last descriptor (if relevant)</li> <li>• Bit 3: DMA Complete with Error</li> <li>• Bit 4: DMA Complete with more than 4GBytes of data transferred</li> <li>• Bit 5: reserved</li> <li>• Bit 6: DMA successfully stopped by user</li> <li>• Bit 7: DMA incorrectly ended (buffer or descriptor not released)</li> </ul> </li> </ul> <p>Note that if DMA ends because of an error, the Error Status field will be something other than 8'b0. This field is automatically cleared when Bit 0 of Control Field is set to 1b.</p> <ul style="list-style-type: none"> <li>• Bit [15:8]: <b>SRC_ERROR:</b> <ul style="list-style-type: none"> <li>• Bit 0: Completion Timeout</li> <li>• Bit 1: CA received if on PCIe domain, EXOKAY received if on AXI domain</li> <li>• Bit 2: EP or ECRC received if on PCIe domain, SLVERR response received if on AXI domain</li> <li>• Bit 3: UR received PCIe if on PCIe domain, DECERR response received if on AXI domain</li> <li>• Bits 4-7: reserved</li> </ul> </li> </ul> <p>These Fields are automatically cleared when Bit 0 of the Control field is set to 1b.</p> <ul style="list-style-type: none"> <li>• Bit [23:16]: <b>DEST_ERROR:</b> as for SRC_ERROR</li> <li>• Bit [31:24]: reserved</li> </ul>
<b>DMA_PRC_LENGTH</b>	0x24 – 0x27	RO	<p><b>DMA_PRC_LENGTH:</b> this 32-bit register provides the amount of data in bytes actually transferred from the Source to the Destination. It is only relevant if Bit 4 of the STATUS Field is cleared.</p>
--	0x27 - 0x3F	RO	Reserved

Table 29: DMA Engine Configuration Space Registers

## 14.6 Address Translation Registers

The Address Translation registers are split into six sections, depending on the Internal Bus Slave port to translate.

- The PCI Express Address Translator uses eight Address Translation Tables per BAR.
- Each AXI4 Slave Address Translator uses eight Address Translation Tables.

Byte Address	R/W	Description
0x0600 - 0x06FF	RO or RW	ATR_PCIE_WIN0: PCIe Window 0 Address Translation Tables 0-7
0x0700 - 0x07FF	RO or RW	ATR_PCIE_WIN1: PCIe Window 1 Address Translation Tables 0-7
0x0800 - 0x08FF	RO or RW	ATR_AXI4_SLV0: AXI4 Slave 0 Address Translation Tables 0-7
0x0900 - 0x09FF	RO or RW	ATR_AXI4_SLV1: AXI4 Slave 1 Address Translation Tables 0-7
0x0A00 - 0x0AFF	RO or RW	ATR_AXI4_SLV2: AXI4 Slave 2 Address Translation Tables 0-7
0x0B00 - 0x0BFF	RO or RW	ATR_AXI4_SLV3: AXI4 Slave 3 Address Translation Tables 0-7

**Table 30: Address Translation Registers**

When these Address Translation Tables are not hardwired by Core Constants, they are read/write and their default values after reset are {32'h0, 32'h0, 32'h0, 32'h0, 32'h0, 32'h00000016}.

Each Address Translation Table is 32 bytes in length, and is described in the following table:

Name	Byte Address	R/W	Description
<b>SRC_ADDR [31:12]</b> <b>ATR_PARAM</b>	0x00 - 0x03	RO or RW	<ul style="list-style-type: none"> <li>• Bit [0]: <b>ATR_IMPL</b>: ATR_IMPL Field is 1 bit long. When set to 1, it indicates that the Translation Address Table is implemented.</li> <li>• Bit [6:1]: <b>ATR_SIZE</b>: ATR_SIZE is 6 bits long and defines the Address Translation Space Size. This space size in bytes is equal to <math>2^{(ATR\_SIZE + 1)}</math>. Allowed values for this field are from 6'd11 (<math>2^{12} = 4</math> KBytes) to 6'd63 (<math>2^{64} = 16</math> ExaBytes) only.</li> <li>• Bit [11:7]: reserved</li> <li>• Bit [31:12]: <b>SRC_ADDR</b> [31:12]</li> </ul>
<b>SRC_ADDR [63:32]</b>	0x04 - 0x07	RO or RW	<b>SRC_ADDR</b> [63:32] defines the starting address of the address translation space.
<b>TRSL_ADDR [31:12]</b>	0x08 - 0x0B	RO or RW	<b>TRSL_ADDR</b> defines the starting translated address of the address translation space.
<b>TRSL_ADDR [63:32]</b>	0x0C - 0x0F	RO or RW	<p>SRC_ADDR and TRSL_ADDR are aligned on the Address Translation Space Size. Therefore, SRC_ADDR [integer(ATR_SIZE):0] and TRSL_ADDR [integer(ATR_SIZE):0] are ignored.</p> <p><b>Note:</b> TRSL_ADDR[11:0:] is currently reserved.</p>

**Table 31: Address Translation Table Registers**



Name	Byte Address	R/W	Description
TRSL_PARAM	0x10 - 0x13	RO or RW	<ul style="list-style-type: none"> <li>Bit [3:0]: <b>TRSL_ID</b> is 4 bits long and defines the Translated ID of the request. The Completer ID Field of a read or write request to an address that targets the specified translation space will be converted to a TRSL_ID value. Allowed values for this field are: <ul style="list-style-type: none"> <li>4'd0: PCIe Tx/Rx Interface</li> <li>4'd1: PCIe Config Interface (only possible when XR3AXI_ATR_MIXED_INTERFACE[1] is set to 1)</li> <li>4'd2: AXI4-Lite Master Interface (External Registers) (only possible when XR3AXI_ATR_MIXED_INTERFACE[2] is set to 1)</li> <li>4'd4 + k: AXI4 Master Number <b>k=0..3</b> Interface</li> <li>4'd8 + l: AXI4 Stream Number <b>l=0..3</b> Interface</li> <li>4'd12: Bridge Internal Registers (only possible when XR3AXI_ATR_MIXED_INTERFACE[0] is set to 1)</li> </ul> </li> <li>Bit [15:4]: reserved</li> <li>Bit [27:16]: <b>TRSF_PARAM</b> is 12 bits long and provides the Translated Parameter of the request. The Transfer Parameter field of a read or write request to an address that targets this address translation space will be converted to the TRSF_PARAMETER value.</li> <li>Bit [31:28]: reserved</li> </ul>
--	0x14 - 0x17	RO	reserved
TRSL_MASK	0x18 - 0x1F	RO	<p><b>TRSL_MASK</b> defines the translation table mask address. It is equal to (0 - Table Size), where the table size is equal to <math>2^{(ATR\_SIZE + 1)}</math>. For example, if the table size is fixed to 256 KBytes, TRSL_MASK is equal to (0 - 256KBytes) = 64'hFFFFFFFFC0000.</p>

Table 31: Address Translation Table Registers

**Note:** If an address translation event occurs, it is reported to ISTATUS\_X\_ADT\_X registers and so to the ISTATUS\_HOST and ISTATUS\_LOCAL registers (see ISTATUS and IMASK registers). You can specify whether an interrupt is issued to the host and/or to the local processor by configuring IMASK\_HOST and IMASK\_LOCAL.

## 14.7 PCI Express Configuration Space

When accessing the Bridge Configuration Space at addresses 0x1000 - 0x1FFF, the accesses are routed to the PCIe Controller Backend Configuration Space Interface. This enables a Local processor on the AXI domain to read from or write to the PCIe Controller Configuration Space (see [Appendix A: Register Content of the PCIe Config Space](#) ).

## 14.8 Bridge External Registers

When accessing the Bridge Configuration Space at addresses 0x2000-0x3FFF, the accesses are routed to the AXI4-Lite Master Interface. This enables a Host processor on the PCIe domain to read from or write to the configuration space of modules located on the AXI domain.

**Note:** The External Registers module, included in the Core Reference Design, provides examples of how to implement R/W registers and how to connect an AXI4-Lite Master Interface to several AXI4-Lite Slave Interfaces. See the QuickPCIe Expert Getting Started for more information.

## Appendix A: Register Content of the PCIe Config Space

This Appendix describes the layout of the PCI Express Configuration Space and provides the mapping for each register in the Space. It also describes how each register is implemented in the QuickPCIe Expert Core, and highlights any differences between the description of the register in the relevant specification, and the implementation of that register in the Core.

### A.1 PCI Configuration Space

The following table describes the layout of the PCI Configuration Space:

Offset	Description
000h ... 03Ch	Type0 (Endpoint) Standard PCI configuration header
040h ... 07Ch	reserved
080h ... 0B8h	PCI Express Capability
0BCh ... 0CCh	reserved
0D0h ... 0D8h	reserved
0DCh	reserved
0E0h ... 0F4h	MSI Capability (0F0h...0F7h reserved for future implementation of per-vector masking)
0F8h ... 0FCh	PCI Power Management Capability

**Table 32: PCI Configuration Space Layout**

### A.1.1 Type 0 Standard PCI Configuration Header

Table 33: Type 0 Standard PCI Configuration Header

31:24	23:16	15:8	7:0	Byte Offset
Device ID		Vendor ID		000h
Status		Command		004h
Class Code			Revision ID	008h
BIST	Header Type	Latency Timer	Cache Line Size	00Ch
Base Address 0				010h
Base Address 1				014h
Base Address 2				018h
Base Address 3				01Ch
Base Address 4				020h
Base Address 5				024h
				028h
Subsystem ID		Subsystem Vendor ID		02Ch
Expansion ROM base address				030h
			Capabilities PTR	034h
				038h
		Int. Pin	Int. Line	03Ch

### A.1.2 PCI Express Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
Capabilities Register		Next Cap PTR	Capability ID	080h
Device Capabilities				084h
Device Status		Device Control		088h
Link Capabilities				08Ch
Link Status		Link Control		090h
Device Capabilities 2				0A4h
Device Status 2		Device Control 2		0A8h
Link Capabilities 2				0ACh
Link Status 2		Link Control 2		0B0h

Table 34: PCI Express Capability Structure

### A.1.2.1 PCI Express Capability List Register (Offset 080h)

The following table provides additional information about the implementation of this register in the QuickPCle Expert Core. For a full description, see the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
7:0	Capability ID	Always 10h.	RO
15:8	Next Capability Pointer	Depends on which capabilities are implemented.	RO

Table 35: PCI Express Capability List Register

### A.1.2.2 PCI Express Capabilities Register (Offset 082h)

The following table provides additional information about the implementation of this register in the QuickPCle Expert Core. For a full description, see the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
3:0	Capability Version	Hardwired to 2h.	RO
7:4	Device/Port Type	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
13:9	Interrupt Message Number	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
14	Reserved	Hardwired to 0	RO
15	Undefined	Hardwired to 0	-

Table 36: PCI Express Capabilities Register

### A.1.2.3 Device Capabilities Register (Offset 084h)

The following table provides additional information about the implementation of this register in the QuickPCle Expert Core. For a full description, see the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
2:0	Max_Payload_Size_Supported	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
4:3	Phantom Functions Supported	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
5	Extended Tag Field Supported	Hardwired to 1: 8-bit tag field.	RO
8:6	Endpoint L0s Acceptable Latency	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
11:9	Endpoint L1 Acceptable Latency	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
14:12	Undefined	Hardwired to 000.	RO
15	Role-Based Error Reporting	Hardwired to 1.	RO
17:16	Reserved	Hardwired to 00.	RO

Table 37: Device Capabilities Register

Bits	Field	Additional Description	Attr.
25:18	Captured Slot Power Limit Value	--	RO
27:26	Captured Slot Power Limit Scale	--	RO
28	Function Level Reset Capability	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO

Table 37: Device Capabilities Register

#### A.1.2.4 Device Control Register (Offset 088h)

This register is implemented as defined by the *PCI Express Specification 3.0*.

#### A.1.2.5 Device Status Register (Offset 08Ah)

The following table provides additional information about the implementation of certain bits of this register in the QuickPCle Expert Core.

All other bits of this register are implemented as defined by the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
4	AUX Power Detected	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
5	Transactions Pending	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO

Table 38: Device Status Register

#### A.1.2.6 Link Capabilities Register (Offset 08Ch)

The following table provides additional information about the implementation of this register in the QuickPCle Expert Core. For a full description, see the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
3:0	Max Link Speed	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
9:4	Maximum Link Width	Defined by QPCIE_.	RO
11:10	ASPM Support	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
14:12	L0s Exit Latency		RO
17:15	L1 Exit Latency		RO
18	Clock Power Management		RO
19	Surprise Down Error Reporting Capable		RO
20	Data Link Layer Active Reporting Capable		RO
21	Link Bandwidth Notification Capability	<ul style="list-style-type: none"> <li>• 0 for upstream ports.</li> <li>• 1 for downstream ports.</li> </ul>	RO

Table 39: Link Capabilities Register

Bits	Field	Additional Description	Attr.
22	ASPM Optionality Compliance	Set to 1b.	HwInit
23	reserved	--	--
31:24	Port Number	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	HwInit

Table 39: Link Capabilities Register

### A.1.2.7 Link Control Register (Offset 090h)

This register is implemented as defined by the *PCI Express Specification 3.0*.

### A.1.2.8 Link Status Register (Offset 092h)

The following table provides additional information about the implementation of certain bits of this register in the QuickPCIe Expert Core.

All other bits of this register are implemented as defined by the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
12	Slot Clock Configuration	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	HwInit

Table 40: Link Status Register

### A.1.2.9 Device Capabilities Register (Offset 0A4h)

The following table provides additional information about the implementation of this register in the QuickPCIe Expert Core. For more information, see the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
3:0	Completion Timeout Ranges Supported	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	HwInit
4	Completion Timeout Disable Supported		RO
5	ARI Forwarding Supported	Hardwired to 0.	RO
6	AtomicOp Routing Supported	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
7	32-Bit AtomicOp Completer Supported		RO
8	64-Bit AtomicOp Completer Supported		RO
9	128-Bit CAS Completer Supported		RO

Table 41: Device Capabilities Register

Bits	Field	Additional Description	Attr.
10	No RO-enabled PR-PR Passing	Hardwired to 0.	HwInit
11	LTR Mechanism Supported		RO
13:12	TPH Completer Support		RO
19:18	OBFF Supported		RO
20	Extended FMT Field Supported		RO
21	End_End TLP Prefix Supported		HwInit
23:22	Max End_End TLP Prefixes		HwInit

Table 41: Device Capabilities Register

#### A.1.2.10 Device Control 2 Register (Offset 0A8h)

The following table provides additional information about the implementation of certain bits of this register in the QuickPCle Expert Core.

All other bits of this register are implemented as defined by the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
5	ARI Forwarding Enable	Hardwired to 0.	RW
8	IDO Request Enable		RW
9	IDO Completion Enable		RW
10	LTR Mechanism Enable		RW/RsvdP
14:13	OBFF Enable		RW/RsvdP
15	End-End TLP Prefix Blocking		RW

Table 42: Device Control 2 Register

#### A.1.2.11 Device Status 2 Register (Offset 0AAh)

This register is implemented as defined by the *PCI Express Specification 3.0*.

#### A.1.2.12 Link Capabilities 2 Register (Offset 0ACh)

The following table provides additional information about the implementation of this register in the QuickPCle Expert Core. For more information, see the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
0	reserved	Hardwired to 0.	--
7:1	Supported Link Speeds Vector	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
8	Crosslink Supported	Hardwired to 0.	RO
31:9	reserved	Hardwired to 0.	--

Table 43: Link Capabilities 2 Register

### A.1.2.13 Link Control 2 Register (Offset 0B0h)

This register is implemented as defined by the *PCI Express Specification 3.0*.

### A.1.2.14 Link Status 2 Register (Offset 0B2h)

This register is implemented as defined by the *PCI Express Specification 3.0*.

## A.1.3 MSI Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
Message Control		Next Pointer	Cap ID	0E0h
Message Address				0E4h
Message Upper Address				0E8h
			Message Data	0F0h

Table 44: Message Signaled Interrupt (MSI) Capability Structure

### A.1.3.1 Capability ID for MSI

The following table provides additional information about the implementation of this register in the QuickPCIe Expert Core. For a full description, see the *PCI Local Bus Specification 3.0*.

Bits	Field	Additional Description	Attr.
7:0	Capability ID	Always 05h.	RO

Table 45: Capability ID for MSI

### A.1.3.2 Next Pointer for MSI

The following table provides additional information about the implementation of this register in the QuickPCIe Expert Core. For a full description, see the *PCI Local Bus Specification 3.0*.

Bits	Field	Additional Description	Attr.
7:0	Next Pointer	Depends on which capabilities are implemented.	RO

Table 46: Next Pointer for MSI

### A.1.3.3 Message Control for MSI

The following table provides additional information about the implementation of certain bits of this register in the QuickPCIe Expert Core.



All other bits of this register are implemented as defined by the *PCI Local Bus Specification 3.0*.

Bits	Field	Additional Description	Attr.
3:1	Multiple Message Capable	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
7	64-Bit Address Capable	Hardwired to 1.	RO
8	Per-Vector Masking Capable	Hardwired to 0.	
15:9	reserved	Hardwired to 0.	--

**Table 47: Message Control for MSI**

#### A.1.3.4 Message Address for MSI

This register is implemented as defined by the *PCI Local Bus Specification 3.0*.

#### A.1.3.5 Message Upper Address for MSI

This register is implemented as defined by the *PCI Local Bus Specification 3.0*.

#### A.1.3.6 Message Data for MSI

This register is implemented as defined by the *PCI Local Bus Specification 3.0*.

## A.1.4 Power Management Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
Power Management Capabilities		Next Item PTR	Cap ID	0F8h
Data	PMCSR_BSE Bridge Support Extensions	Power Management Control & Status Register		0FCh

Table 48: Power Management Capability Structure

### A.1.4.1 Capability ID

The following table provides additional information about the implementation of this register in the QuickPCIe Expert Core. For a full description, see the *PCI Bus Power Management Interface Specification 1.2*.

Bits	Field	Additional Description	Attr.
7:0	Cap_ID	Always 01h.	RO

Table 49: Capability ID

### A.1.4.2 Next Item Pointer

The following table provides additional information about the implementation of this register in the QuickPCIe Expert Core. For a full description, see the *PCI Bus Power Management Interface Specification 1.2*.

Bits	Field	Additional Description	Attr.
7:0	Next Pointer	Depends on which capabilities are implemented.	RO

Table 50: Next Item Pointer

### A.1.4.3 Power Management Capabilities

The following table provides additional information about the implementation of this register in the QuickPCIe Expert Core. For a full description, see the *PCI Bus Power Management Interface Specification 1.2*.

Bits	Field	Additional Description	Attr.
15:11	PME_Support	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
10	D2_Support		RO
9	D1_Support		RO
8:6	Aux_Current		RO
5	DSI		RO
4	reserved	Hardwired to 0.	RO
3	PME Clock		RO
2:0	Version	Set to 011b.	RO

Table 51: Power Management Capabilities

#### A.1.4.4 Power Management Control/Status (PMCSR)

The following table provides additional information about the implementation of certain bits of this register in the QuickPCle Expert Core.

All other bits of this register are implemented as defined by the *PCI Bus Power Management Interface Specification 1.2*.

Bits	Field	Additional Description	Attr.
14:13	Data_Scale	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO
3	No_Soft_Reset	Hardwired to 1.	RO

**Table 52: PMCSR**

#### A.1.4.5 PMCSR\_BSE

All bits of this register are hardwired to 0.

#### A.1.4.6 Data

The following table provides additional information about the implementation of this register in the QuickPCle Expert Core.

Bits	Field	Additional Description	Attr.
7:0	Data_Register	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO

**Table 53: Data Register**

## A.2 PCI Express Extended Configuration Space

The following table describes the layout of the PCI Express Extended Configuration Space:

Offset	Description
100h ... 104h	Vendor-specific capability with VSECID=1556h; RevID=1h
120h ... 124h	reserved
130h ... 13Ch	reserved
140h ... 17Ch	reserved
200h ... 234h	Advanced Error Reporting capability (optional)
300h ... 328h	Secondary PCI Express extended capability (implemented in Function 0 if the device supports speeds of 8 Gbps)
400h ... 428h	reserved
800h ... FFCh	reserved

**Table 54: PCI Express Extended Configuration Space Layout**

## A.2.1 Vendor Specific Extended Capability Structure

31:24	23:16	15:8	7:0	Byte Offset
Vendor-Specific Extended Capability Header				100h
Vendor-Specific Header				104h
Vendor-Specific Registers				108h

**Table 55: Vendor-Specific Extended Capability Structure**

### A.2.1.1 Vendor-Specific Extended Capability Header

The following table provides additional information about the implementation of this register in the QuickPCle Expert Core. For a full description, see the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 000Bh.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

**Table 56: Vendor-Specific Extended Capability Header**

### A.2.1.2 Vendor-Specific Header

The following table provides additional information about the implementation of this register in the QuickPCle Expert Core. For a full description, see the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
15:0	VSEC ID	Always 1556h.	RO
19:16	VSEC Rev	Hardwired to 1h.	RO
31:20	VSEC Length	Hardwired to 008h.	RO

**Table 57: Vendor-Specific Header**

## A.2.2 Advanced Error Reporting Capability Structure

The following table shows the Advanced Error Reporting Extended Capability Structure for Function 0. For Functions 1 - 7, the byte offset is from 100h to 134h.

31:24	23:16	15:8	7:0	Byte Offset
PCI Express Enhanced Capability Header				800h
Uncorrectable Error Status Register				804h
Uncorrectable Error Mask Register				808h
Uncorrectable Error Severity Register				80Ch
Correctable Error Status Register				810h
Correctable Error Mask Register				814h
Advanced Error Capabilities and Control Register				818h
Header Log Register				81Ch
Root Error Command				82Ch
Root Error Status				830h
Error Source Identification Register				834h

**Table 58: Advanced Error Reporting Extended Capability Structure**

### A.2.2.1 Advanced Error Reporting Extended Capability Header

The following table provides additional information about the implementation of this register in the QuickPCIe Expert Core. For a full description, see the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 0001h.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

**Table 59: AER Extended Capability Header**

### A.2.2.2 Uncorrectable Error Status Register

This register is implemented as defined by the *PCI Express Specification 3.0*.  
Bits 21, 22, 23, and 25 are hardwired to 0.

### A.2.2.3 Uncorrectable Error Mask Register

This register is implemented as defined by the *PCI Express Specification 3.0*.  
Bits 20, 21, 22, and 24 are hardwired to 0.

### A.2.2.4 Uncorrectable Error Severity Register

This register is implemented as defined by the *PCI Express Specification 3.0*.  
Bits 21, 22, 23, and 25 are hardwired to 0.

### A.2.2.5 Correctable Error Status Register

This register is implemented as defined by the *PCI Express Specification 3.0*.

Bits 14 and 15 are hardwired to 0.

### A.2.2.6 Correctable Error Mask Register

This register is implemented as defined by the *PCI Express Specification 3.0*.

Bits 14 and 15 are hardwired to 0.

### A.2.2.7 Advanced Error Capabilities and Control Register

The following table provides additional information about the implementation of certain bits of this register in the QuickPCIe Expert Core.

All other bits of this register are implemented as defined by the *PCI Bus Power Management Interface Specification 1.2*.

Bits 9 - 11 are hardwired to 0.

Bits	Field	Additional Description	Attr.
5	ECRC Generation capable	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RWS
7	ECRC Check Capable		RO

Table 60: Advanced Error Capabilities and Control Register

### A.2.2.8 Header Log Register

This register is implemented as defined by the *PCI Express Specification 3.0*.

### A.2.2.9 Root Error Command Register

This register is implemented as defined by the *PCI Express Specification 3.0*.

### A.2.2.10 Root Error Status Register

The following table provides additional information about the implementation of certain bits of this register in the QuickPCIe Expert Core.

All other bits of this register are implemented as defined by the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
31:27	Advanced Error Interrupt Message Number	Defined by Core Parameters (see <a href="#">Chapter 5</a> ) or Bridge Internal Registers (see <a href="#">Chapter 14</a> ).	RO

Table 61: Root Error Status Register

### A.2.2.11 Error Source Identification Register

This register is implemented as defined by the *PCI Express Specification 3.0*.

## A.2.3 Secondary PCI Express Extended Capability Structure

The following table shows the Advanced Error Reporting Extended Capability Structure for Function 0. For Functions 1 - 7, the byte offset is from 100h to 134h.

31:24	23:16	15:8	7:0	Byte Offset
Secondary PCI Express Extended Capability Header				300h

**Table 62: Secondary PCI Express Extended Capability Structure**

### A.2.3.1 Secondary PCI Express Extended Capability Header

The following table provides additional information about the implementation of this register in the QuickPCIe Expert Core. For a full description, see the *PCI Express Specification 3.0*.

Bits	Field	Additional Description	Attr.
15:0	PCI Express Extended Capability ID	Always 0019h.	RO
19:16	Capability Version	Hardwired to 1h.	RO
31:20	Next Capability Offset	Depends on which capabilities are implemented.	RO

**Table 63: Secondary PCI Express Extended Capability Header**



## Appendix B: Message and Event Handling

### B.1 Sending Messages to the PCI Express Interface

To send a Message through the Address Translation Tables to the PCIe interface, configure the Address Translation Table with:

- TRSL\_PARAM[3:0] = 4'd0 to target PCIe Transmit Interface
- TRSL\_PARAM[18:16] = TRSF\_PARAM[2:0] = 3'b100 to send Message Type

Then, send a Write Request that targets this Table, with:

- a Write request address of 12 LSB bits equal to 12'b0.
- a Payload equal to a full Message: including the 4DW Header, which specifies the Message Code and Routing information.

To send a Message through the DMA to the PCIe interface, configure a DMA with:

- DMA\_DESTPARAM[3:0] = DEST\_ID = 4'd0 to target PCIe Transmit Interface
- DMA\_DESTPARAM[18:16] = TRSF\_PARAM[2:0] = 3'b100 to send Message Type
- DMA\_DESTADDR[63:0] = 64'h0
- DMA\_LENGTH should be equal to the full Message Size including the 4DW Header

Then launch the DMA with the Data Payload to transfer equal to the full Message, including the 4DW Header, which specifies the Message Code and Routing information.

### B.2 Received Message Processing

The following table describes how PCI Express messages are handled by the QuickPCle Expert Core.

For more information about each of these message types, see the *PCI Express™ Base Specification Revision 3.0*.

Received Message	Endpoint
Unlock	Unsupported by QuickPCle Expert; silently discarded
Invalidate Request (ATS)	Unsupported by QuickPCle Expert; discarded
Invalidate Completion (ATS)	Unsupported by Endpoint; discarded
Page Request (ATS)	Unsupported by Endpoint; discarded
PRG Response (ATS)	Unsupported by QuickPCle Expert; discarded
Latency Tolerance Reporting	Unsupported by QuickPCle Expert; discarded
Optimized Buffer Flush/Fill	Unsupported by QuickPCle Expert; discarded
PM_Active_State_Nak	Processed internally. If the QuickPCle Expert receives this message while it is preparing for transition to ASPM L1, it stops the transition.

**Table 64: Received Message Processing**

Received Message	Endpoint
PM_PME	Unsupported by Endpoint; discarded
PME_Turn_Off	Processed internally. Causes the QuickPCIe Expert to enter L2 state and the link to be turned off.
PME_TO_Ack	Unsupported by Endpoint; discarded
Assert_INTA/B/C/D	Treated as malformed by Endpoint; discarded
Deassert_INTA/B/C/D	
ERR_COR	Unsupported by Endpoint; discarded
ERR_NONFATAL	
ERR_FATAL	
Ignored messages	Silently discarded
Set_Slot_Power_Limit	Processed internally. Values are captured in the configuration space.
Vendor Defined Type 0	Unsupported by QuickPCIe Expert; discarded
Vendor Defined Type 1	Silently discarded by QuickPCIe Expert

Table 64: Received Message Processing

## B.3 Event Processing

The following table describes how PCI Express events are processed in the QuickPCIe Expert Core, and the actions taken by both QuickPCIe Expert and by the application.

Event	QuickPCIe Expert Action	Application Action
Power Management Event received (Rootport only)	Captured in PCIe Capability Root Status Register. The PME# interrupt is logged to the QuickPCIe Expert Local Processor Interrupt Status Register and reported to the application by interrupt_out[30], if the corresponding interrupt's source is activated.	Processing is application-specific. Note that PME Interrupt Enable must be set in the PCIe Root Control Register to enable an interrupt. We recommend that the application checks and then clears the PCIe Status Control Register after each interrupt.
Legacy interrupt received (Rootport only)	Legacy Interrupts are logged to the QuickPCIe Expert Local Processor Interrupt Status Register and reported to the application by interrupt_out[27..24], if the corresponding interrupt's sources are activated.	Processing is application-specific.
MSI received (Rootport only)	MSI are standard MemWr TLP targeting the MSI Capture Address. When an MSI is captured by the Core, it is logged to the QuickPCIe Expert Local Processor Interrupt Status and MSI Message Registers and reported to the application by interrupt_out[28], if corresponding interrupt's source is activated.	Processing is application-specific.
AER error (Rootport only)	AER Error Interrupts are logged to the QuickPCIe Expert Local Processor Interrupt Status Register and reported to the application by interrupt_out[29,] if the corresponding interrupt's source is activated.	Processing is application-specific.
System Error (Rootport only)	System Error Interrupts are logged to the QuickPCIe Expert Local Processor Interrupt Status Register and reported to the application by interrupt_out[32,] if the corresponding interrupt's source is activated.	Processing is application-specific.
Hotplug event (Rootport only)	Hotplug events are logged to the QuickPCIe Expert Local Processor Interrupt Status Register and reported to the application by interrupt_out[30,] if the corresponding interrupt's source is activated.	Processing is application-specific.
Equalization Request (Rootport only)	No action required. Request is ignored. The application is not informed.	
Link partner requests L1 entry (can only be initiated by Upstream Ports)	For Downstream Ports only: The Core enters L1 as soon as it's ready.	
Link partner requests L2 entry (can only be initiated by Downstream Ports)	No action required. Causes the Core to enter L2 state and the link to be turned off. The application is not informed.	

**Table 65: Event Processing**

Event	QuickPCIe Expert Action	Application Action
Link partner enters ASPM L0s	No action required. Causes the Core to enter L0s. The application is not informed.	
Link partner requests ASPM L1 entry (can only be initiated by Upstream Ports)	No action required. If ASPM L1 is not enabled then the Core refuses ASPM L1 entry, otherwise it waits until the transmit buffer is empty and enters ASPM L1. The application is not informed.	
Legacy Power Management state change	No action required. In D1/D2/D3 hot/cold: the Core is not allowed to transmit interrupts and TLPs from the application. The PM state change event is logged to the QuickPCIe Expert Local Processor Interrupt Status Register and reported to the application by <code>interrupt_out[30]</code> , if the corresponding interrupt's source is activated.	Processing is application-specific. The application can check the QuickPCIe Expert PCI Legacy Power Management State Register to verify the PCIe power state.
Link exits L2 state	QuickPCIe Expert asserts <code>pl_rstn_srst_out</code> and <code>tl_crstn_crst_out</code> (low) in order to reset all Core logic, except sticky registers.	The application should assert <code>pl_rstn/tl_rstn/tl_crstn</code> (low) or <code>pl_srst/tl_srst/tl_crst</code> (high) for at least one clock cycle. Resetting the application logic is optional.
PCIe commanded Hot Reset	QuickPCIe Expert asserts low <code>pl_rstn_srst_out</code> in order to reset all Core logic, except sticky registers.	The application should assert <code>pl_rstn/tl_rstn</code> (low) or <code>pl_srst/tl_srst</code> (high) for at least one clock cycle. Resetting the application logic is optional.
Data link goes down	When the Core is an Endpoint: QuickPCIe Expert asserts <code>pl_rstn_srst_out</code> and <code>tl_crstn_crst_out</code> (low) in order to reset all Core logic, except sticky registers. When the Core is Rootport: QuickPCIe Expert asserts <code>pl_rstn_srst_out</code> (low) in order to reset all Core logic, except sticky registers and configuration registers.	When the Core is Endpoint: the application should assert <code>pl_rstn/tl_rstn/tl_crstn</code> (low) or <code>pl_srst/tl_srst/tl_crst</code> (high) for at least one clock cycle. When the Core is Rootport: the application should assert <code>pl_rstn/tl_rstn</code> (low) or <code>pl_srst/tl_srst</code> (high) for at least one clock cycle. Resetting the application logic is optional.

Table 65: Event Processing

## Appendix C: AXI Interface Implementation

### C.1 AXI Implementation Notes

The following table provides further information about how certain functionality described in the *AMBA® AXI™ and ACE™ Protocol Specification* is implemented in the QuickPCle Expert Core. The relevant sections of the specification are provided for further information.

Interface	Spec Section	Comments
AXI3, AXI4	A4: Transaction Attributes	AXI Slave attributes (AxCACHE) and access permissions (AxPROT) are ignored since they are not applicable to QuickPCle Expert. All Slave transactions are handled as Device Non-bufferable transactions. In particular, ordering is preserved for any set of AXI Slave transactions, regardless of the transaction attributes. However, DMA and Address Translation enable you to set AXI Master attributes and access permissions based on your requirements.
AXI3, AXI4	A5.3.6: Width of transaction ID fields	The AXI Master Transaction ID fields are 4 bits wide. AXI Slave Transaction ID fields are 4 bits wide, but may be extended to the value of QPCIE_AXI_SLV0-3_ID_WIDTH when this constant is defined (see <a href="#">Section 5.1</a> ).
AXI3 Slave, AXI4 Slave	A5.3: Transaction ordering	The data transfers for a sequence of read transactions are returned in the order in which the master issued the addresses, even if they have different ARIDs. AXI Slave Read data reordering depth is configurable from 1 to 16, regardless of the transaction ID width.
AXI3 Slave, AXI4 Slave	A6.3: Interconnect ordering	AMBA AXI Specification states that when an ordering relationship is required between two read and write transactions, then a master must wait to receive a response to the first transaction before issuing the second transaction. Because MWRs are posted, according to PCIe protocol, the QuickPCle Expert issues write transaction responses before the transaction has reached its final destination. However, since PCIe ordering rules ensure that a subsequent MRD won't pass an MWR, the ordering relationship between read and write transactions is guaranteed.
AXI3 Slave, AXI4 Slave	A6.4: Slave ordering	Responses to multiple read or multiple write transactions are issued in the order in which the transactions arrived, even if they have different IDs.
AXI3 Slave, AXI4 Slave	A6.5: Response before final destination	Because MWRs are posted, according to PCIe protocol, the QuickPCle Expert issues write transaction responses before the transaction has reached its final destination. However, since a write transaction response is issued when the write transaction is submitted to the PCIe controller, the visibility of the transaction to any other transactions from upstream masters is guaranteed.
AXI4	A8.1: QoS signaling	AXI Slave QoS identifiers (AxQOS) are ignored by the QuickPCle Expert. Because AXI ordering rules take precedence over ordering for QoS purposes, QoS to TC/VC translation is not straightforward. However, DMA and Address Translation allow AXI Master QoS identifiers to be set according to user requirements.
AX3, AXI4	A10.3: Default signal values	Master and Slave addresses (AxADDR) are 64-bits wide.

**Table 66: AXI Limitations**

## C.2 AXI Limitations

The following table describes certain differences between the *AMBA® AXI™ and ACE™ Protocol Specification*, and the implementation of the AXI3, AXI4, and AXI4-Lite interfaces in the QuickPCle Expert Core. The relevant sections of the specification are provided for further information.

Interface	Spec Section	Limitations	Comments
AXI3, AXI4	A1.3.1: Data channels	Read and Write Data buses are 64, 128 or 256 bits wide.	8, 16, 32, 512 and 1024 bits data bus widths are not supported.
AXI3 Slave, AXI4 Slave	A3.4.3: Narrow transfers	Narrow transfers are supported when QPCIE_AXI_NARROW_ENABLE is set to 1 (see <a href="#">Section 5.1</a> ).	<ul style="list-style-type: none"> <li>• AXI Master Burst Size (AxBURSTLEN) is: <ul style="list-style-type: none"> <li>• 3'b011 when data bus is 64 bits</li> <li>• 3'b100 when data bus is 128 bits</li> <li>• 3'b101 when data bus is 256 bits</li> </ul> </li> <li><b>Note:</b> The Read address bus (ARADDR) is always aligned on 32-bit boundaries, that is ARADDR[1:0]=2'b0.</li> </ul>
		Otherwise, narrow transfers are not supported.	<ul style="list-style-type: none"> <li>• AXI Slave Burst Size (AxBURSTLEN) is: <ul style="list-style-type: none"> <li>• 3'b011 when data bus is 64 bits</li> <li>• 3'b100 when data bus is 128 bits</li> <li>• 3'b101 when data bus is 256 bits</li> </ul> </li> <li>• AXI Master Burst Size (AxBURSTLEN) is: <ul style="list-style-type: none"> <li>• 3'b011 when data bus is 64 bits</li> <li>• 3'b100 when data bus is 128 bits</li> <li>• 3'b101 when data bus is 256 bits</li> </ul> </li> <li><b>Note:</b> The Read address bus (ARADDR) is always aligned on 32-bit boundaries, that is ARADDR[1:0]=2'b0.</li> </ul>
AXI3	A5.3.3: AXI3 write data interleaving	AXI3 write data interleaving is not supported.	<p>The AXI Slave Write data reordering depth is 1.</p> <p>The AXI Master does not support write interleaving.</p>
AXI3 Slave, AXI4 Slave	A7.2: Exclusive accesses	Exclusive accesses are not supported by Slaves.	<p>The AXI Slave returns xRESP = OKAY to an exclusive access, and updates the memory location, if applicable, regardless of the exclusive access success.</p> <p>However DMA and Address Translation allow the AXI Master lock type to be set according to user requirements.</p> <p><b>Note:</b> the QuickPCle Expert does not check for exclusive access restrictions (see section A7.2.4).</p>
AXI3	A7.3: Locked accesses	Locked Access is not supported.	AWLOCK[1] and ARLOCK[1] signals are not implemented.
AXI3 Master, AXI4 Master	A8.2: Multiple region signaling	AXI Master Region identifiers (AxBURSTID) are tied to 4'h0 in the current QuickPCle Expert version.	AXI Slave Region identifiers (AxBURSTID) are ignored in the current QuickPCle Expert version.
AXI4	A8.3: User-defined signaling	User-defined signaling is not supported.	User signals AWUSER, ARUSER, WUSER, RUSER and BUSER are not implemented.

**Table 67: AXI Limitations**

Interface	Spec Section	Limitations	Comments
AX3, AXI4	A9: Low-power Interface	Low-Power Interface is not supported in the current version of QuickPCle Expert.	CSYSREQ, CSYSACK and CACTIVE signals are not implemented in the current version of QuickPCle Expert. A mechanism that uses the QuickPCle Expert Internal Registers is currently used to enter Low-Power states.
AXI4-Lite	B1.1: Definition of AXI4-Lite	Read and Write Data bus are 32-bits wide.	64-bit data bus widths are not supported. AXI4-Lite Slave makes full use of the write strobes.
ACE, ACE-Lite	C: ACE Protocol Specification	ACE and ACE-Lite Protocols are not supported.	

Table 67: AXI Limitations

### C.3 AXI4-Stream Limitations

The following table describes certain differences between the *AMBA® 4 AXI4-Stream Protocol Specification* and the implementation of AXI4-Streams in the QuickPCle Expert Core. The relevant sections of the specification are provided for further information.

Interface	Spec Section	Limitations	Comments
AXI4-Stream	2.1: Signal list	Data bus is 64, 128 or 256 bits wide.	QuickPCle Expert AXI4 Stream Interfaces do not support all possible integer number of bytes.
AXI4-Stream	2.3.2: Byte types	Position bytes are not supported.	<ul style="list-style-type: none"> <li>Byte qualifier TKEEP of AXI4-Stream In is unused.</li> <li>Byte qualifier TKEEP of AXI4-Stream Out is equal to TSTRB.</li> </ul>
AXI4-Stream In	2.4: Byte qualifiers	Non-Contiguous Strokes are supported when QPCIE_AXI_STI3_PACKING is set to 1 (see <a href="#">Section 5.1</a> ).	--
		Otherwise, Non-Contiguous Strokes are not supported.	For example: <ul style="list-style-type: none"> <li>For a one-beat data transfer, TSTRB can be 16'h03FE, but not 16'h03F6.</li> <li>For a multiple-beat data transfer:               <ul style="list-style-type: none"> <li>On first beat, WSTRB can be 16'hFFC0, but not 16'hF1C0</li> <li>On last AXI Beat, WSTRB can be 16'h07FF, but not 16'h07F1</li> <li>On intermediate AXI Beat, WSTRB should be 16'hFFFF.</li> </ul> </li> </ul>
AXI4-Stream	2.8: User signaling	User signaling is unsupported.	User signal TUSER is not implemented.

Table 68: AXI-Stream Limitations

Interface	Spec Section	Limitations	Comments
AXI4-Stream	2.6: Source and destination signaling	<ul style="list-style-type: none"><li>• TID width is 8-bits.</li><li>• TDEST width is 4-bits.</li><li>• DMA and Address Translation allow AXI4-Stream Out's TID and TDEST to be set at user convenience.</li><li>• AXI4-Stream In uses TID and TDEST to support transfer interleaving, and to route received packets to appropriate DMA or Destination through Address Translation.</li><li>• AXI4-Stream In can support up to 8 fixed TID/TDEST combinations.<ul style="list-style-type: none"><li>• If they are fixed, unexpected TID/TDEST combinations are flushed.</li><li>• If they are not fixed, an unlimited number of TID/TDEST combinations can be supported, but an unexpected TID/TDEST combination can lead to deadlock.</li></ul></li></ul>	

**Table 68: AXI-Stream Limitations**



## Appendix D: PCI Express System Performance

The QuickPCle Expert Core is highly customizable, so choices you make during configuration impact on how the Core functions. Factors affecting system performance include:

- Latency
- Maximum effective bandwidth
- Actual link usage

### D.1 Latency

Latency is the delay in transferring a packet between two points. Global latency, that is, the total latency within a system, is the sum of several factors:

- Data payload size of a TLP
- Core latency between the Application Layer and the Link (varies between Receive and Transmit transactions)
- Switch latency
- Completer latency, such as SDRAM read latency and SRAM mailbox read latency
- PCI Express to PCI/PCI-X bridge latency and latency inherent to a PCI component

You can avoid high global latency by carefully designing your system's architecture (limiting the number of Switches, for example) and by modifying the width of critical Links.

The following table shows typical latency values using a x4 lane for various types of Read Request transactions and for different types of memory (mailbox or SDRAM):

Fabric Environment	Data Payload Size and Latency		
	8B	32B	256B
Link bandwidth usage	64 ns	88 ns	312 ns
Point-to-point, mailbox	240 ns	264 ns	464 ns
Point-to-point, SDRAM	272 ns	296 ns	496 ns
Point-to-point, PCI peripherals	448 ns	472 ns	904 ns
Through switch, mailbox	528 ns	552 ns	776 ns
Through switch, SDRAM	560 ns	584 ns	812 ns
Through switch, PCI peripherals	672 ns	720 ns	1144 ns

**Table 69: Typical latency values of Read Request transactions**

The type of transaction also affects latency:

- **Read transactions (high latency potential):** Most systems generate more than one completion transaction per Read Request. More transactions means greater potential latency.
- **Write transactions (low latency potential):** Write operations are posted and do not require completions. Fewer transactions means lower potential latency.
- **Small transactions (high latency potential):** Many small packets will result in greater latency than one large packet, because data payload transfer time contributes to overall latency.

## D.2 Maximum Effective Bandwidth

Bandwidth is a measure of the rate at which data is transferred at a specific point of a Link. Maximum Effective Bandwidth is the rate at which “valuable” data is transferred at a particular point. It doesn’t include transaction «overhead», such as headers, sequence numbers, CRCs, ECRCs, and other packets like DLLPs and SKIP advanced sets.

**Maximum Effective Bandwidth = data / (data + overhead)**

The table below shows Maximum Effective Bandwidth values for Completion transactions sent in response to consecutive Read or Write requests on a x4 Link. For Read Requests, the Completion transaction size can be split into packets of 64 DW, 32 DW, or 16 DW. An overhead of 5 DW for each packet is presumed, with an additional 2 DW of DLLP for Read Request transactions:

Completion transaction size (bytes / DW)	Completion packet(s) in response to a...			
	Write Request	Read Request 64 DW Packet	Read Request 32 DW Packet	Read Request 16 DW Packet
4 / 1	16%	12%	12%	12%
8 / 2	28%	22%	22%	22%
32 / 8	61%	53%	53%	53%
64 / 16	76%	69%	69%	69%
128 / 32	86%	82%	82%	72%
256 / 64	92%	90%	84%	74%
512 / 128	96%	91%	85%	75%
1024 / 256	98%	92%	86%	75%
2048 / 512	99%	92%	86%	75%
4096 / 1024	99%	93%	86%	76%

**Table 70: Sample Maximum Effective Bandwidth Values**

For example, to calculate the Maximum Effective Bandwidth for a Completion packet of 128 DW, divided into four packets of 32 DW, generated in response to a Read Request:

- 4 = number of Completion Packets
- 5 = number of DW of overhead per packet
- 2 = number of DW per DLLP for the complete Read Request transaction
- **Overhead** = (4 X 5) + 2 = 22
- **Maximum Effective Bandwidth** = 128 / (128 + 22) = 85%

**Note:** A Maximum Payload Size of 64 DW or 128 DW provides between 92% and 96% of effective bandwidth for Write transactions and practically the saturation limit for Read transactions. Maximum payloads in this range can be a good trade-off between latency, congestion, and Core size. Systems that require higher bandwidth might need higher maximum data payloads for effective bandwidth values of up to 99%.

## D.3 Actual Link Usage

Actual Link Usage is a measure of how much a Link is actually used in a given period of time, and is defined by the following equation:

**Actual Link Usage = active time / (active time + idle time)**

## D.4 System Performance Illustrated

You can optimize system performance by defining your objectives (is maintaining a small Core size more important than maximum throughput?) and balancing the following factors:

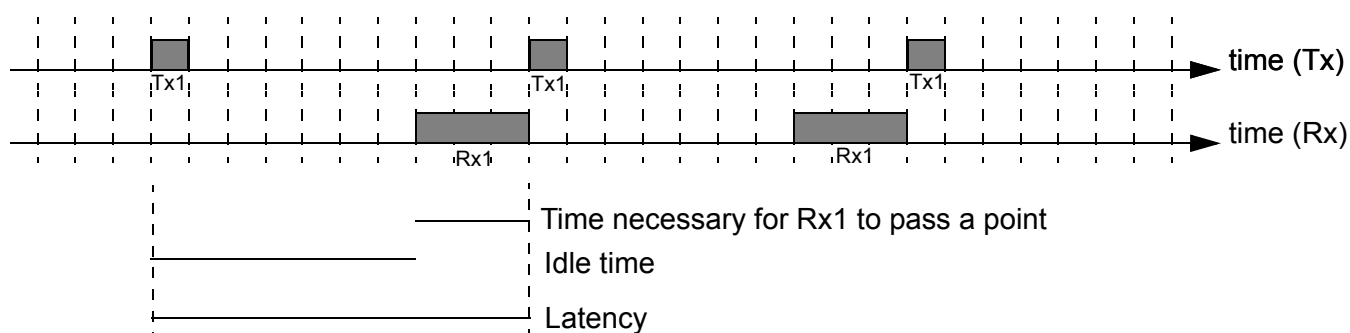
- Maximum packet size
- Global system latency
- Number of outstanding requests per component

*In general, you should consider increasing the number of outstanding requests with small data payloads and in high-latency systems.*

*By default, the maximum value of outstanding requests possible per component is 32, but by setting the Extended tag and the Phantom Function Number Enable bit, you can set outstanding requests up to 2,048. However, once Link Usage has reached 100%, increasing the number of outstanding requests will not improve system performance.*

The following examples use a Maximum Effective Bandwidth of 82%.

### D.4.1 One Outstanding Request, Small Packet



**Figure 15: One outstanding request, small packet**

In this example, Tx1 represents a Read Request of 32 DW. Rx1 represents the corresponding Completion packet. One period (the latency of the transaction) is the time it takes for a Request transaction to be sent to a Completer and have a Completion packet returned. The Actual Link Usage is:

$$\text{Actual Link Usage} = \text{Rx1} / (\text{Rx1} + \text{idle time})$$

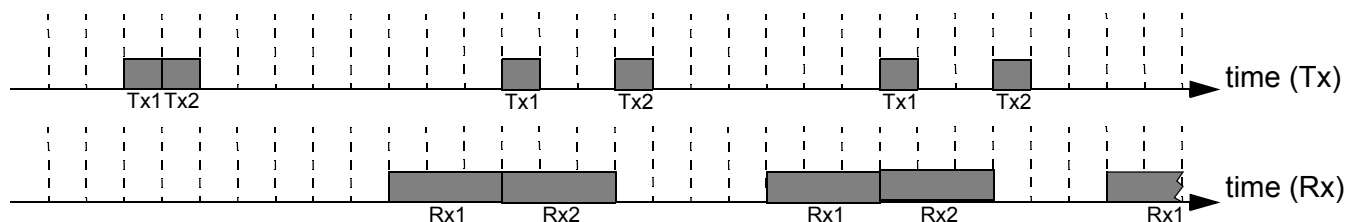
$$\text{Actual Link Usage} = 3 / (3 + 7) = 30\%$$

For a 32 DW Read Request, the Maximum Effective Bandwidth is 82% (see [Table 70](#)). If the actual link usage is 30%, calculate 30% of the Maximum Effective Bandwidth to obtain effective (actual) bandwidth:

$$\text{Effective Bandwidth} = .3 \times .82 = 25\%$$

The effective bandwidth of this Link is only 25%.

### D.4.2 Two Outstanding Requests, Small Packet

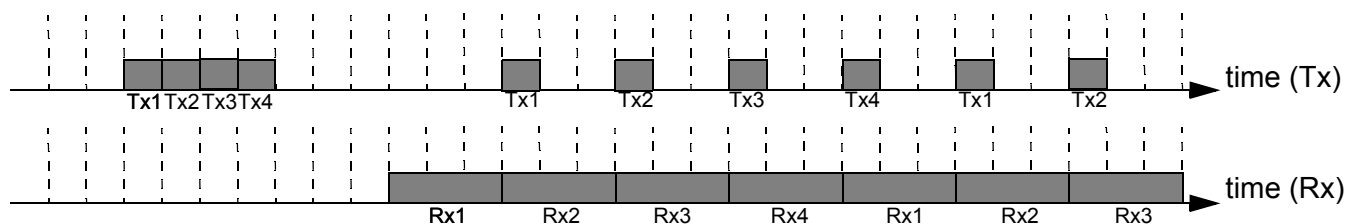


**Figure 16: Two outstanding requests, small packet**

Tx1 and Tx2 represent Read Requests of 32 DW. By increasing the number of outstanding requests to two and leaving all other variables the same, effective bandwidth is:

- Actual Link Usage =  $6 / (6 + 4) = 60\%$
- Effective Bandwidth =  $.6 \times .82 = 49\%$

### D.4.3 Four Outstanding Requests, Small Packet



**Figure 17: Four outstanding requests, small packet**

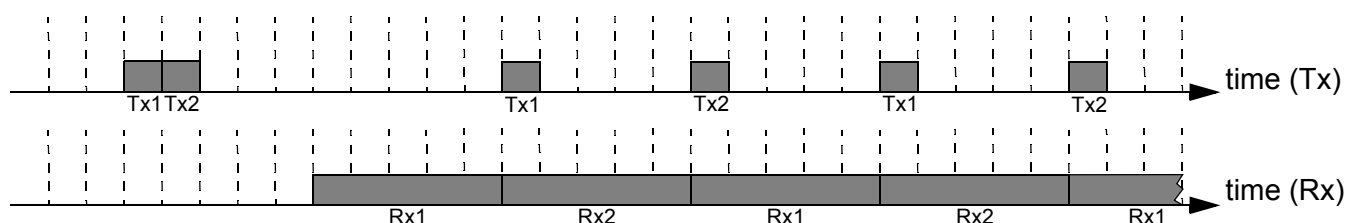
Tx1, Tx2, Tx3, and Tx4 represent Read Requests of 32 DW. Outstanding requests are set to four and the Actual Link Usage = 100%.

Actual Link Usage =  $10 / (10 + 0) = 100\%$

Effective Bandwidth =  $1 \times .82 = 82\%$

This Link is performing at Maximum Effective Bandwidth. Increasing the number of outstanding requests will not improve system performance and will needlessly increase the size of the Core.

### D.4.4 Two Outstanding Requests, Large Packet



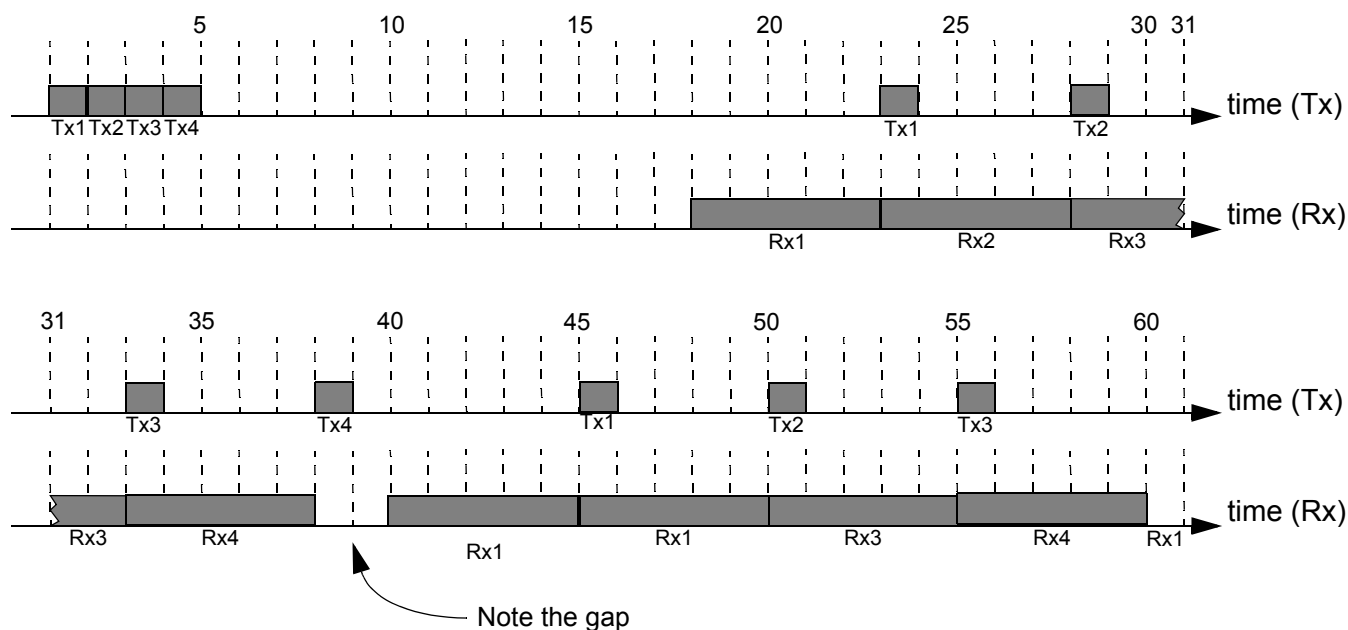
**Figure 18: Two outstanding requests, large packet**

Tx1 and Tx2 represent Read Requests of 32 DW. There are only 2 outstanding requests, but increasing the Maximum Packet Size (Max\_Payload\_Size) enables Maximum Effective Bandwidth.

Actual Link Usage =  $10 / (10 + 0) = 100\%$

Effective Bandwidth =  $1 \times .82 = 82\%$

### D.4.5 High Latency System



**Figure 19: High latency system**

Tx1, Tx2, Tx3, and Tx4 represent Read Requests of 32 DW. The time it takes for a Request to be sent and for the component to receive the corresponding Completion TLP is relatively large, 22 in this case.

Actual Link Usage =  $22 / (22 + 2) = 92\%$

Effective Bandwidth =  $.92 \times .82 = 75\%$

Even with a high number of outstanding requests and large packets (Max\_Payload\_Size), this Link will never reach its Maximum Effective Bandwidth because of high global latency.

## Appendix E: PCI Express Fundamentals

This Appendix describes some of the basic features of the PCI Express protocol for users that are not familiar with the protocol.

For a full description, see the *PCI Express Specifications*.

### E.1 About PCI Express

PCI Express is a third generation, high-bandwidth, low voltage, differential serial interconnect technology that maintains compatibility with existing PCI infrastructures. It is derived from the PCI and PCI-X protocols, however PCI Express represents an architectural leap beyond its sister technologies rather than simply an extension of the existing norms.

PCI Express includes the following features:

- Increased bandwidth: high-speed links can currently transfer up to 8 gigabits per second
- Isochronous transactions: With the introduction of Virtual Channels (VCs) and Traffic Classes (TCs), the designer can prioritize traffic flow
- Serial point-to-point interface
- High bandwidth per pin
- Scalability
- Support for differentiated services, that is, different Qualities of Service (QoS)
- Power Management and budgeting
- Hot-Plug and Hot-Swap support
- Ability to maintain Link-level and end-to-end data integrity
- Advanced Error handling

PCI Express 2.0 and 3.0 increases the interconnect bit rate of PCI Express to offer respectively two and four times higher bandwidth.

High-speed links and faster signaling increase bandwidth from 2.5 Gbps (for PCIe 1.0) to 5 Gbps (for 2.0), then to 8 Gbps (for 3.0). PCI Express 2.0 and 3.0 also create and define several enhancements to the original PCI Express specification, including capability structure expansion, access control services, completion timeout control and virtualization.

PCI Express 2.0 and 3.0 are compatible with all previous versions of PCIe products.

### E.2 PCI Express Lanes and Links

A PCI Express lane is the fundamental connection between two devices. Each lane is defined by a pair of differential transmit signals and a pair of differential receive signals, as illustrated below.

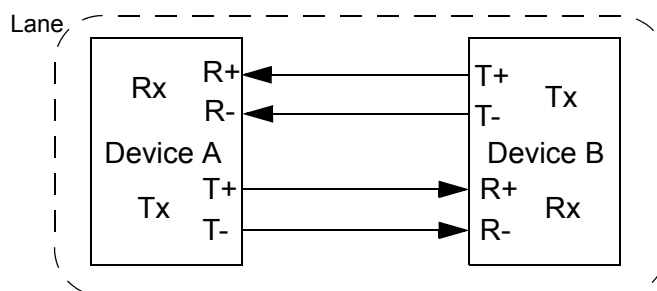
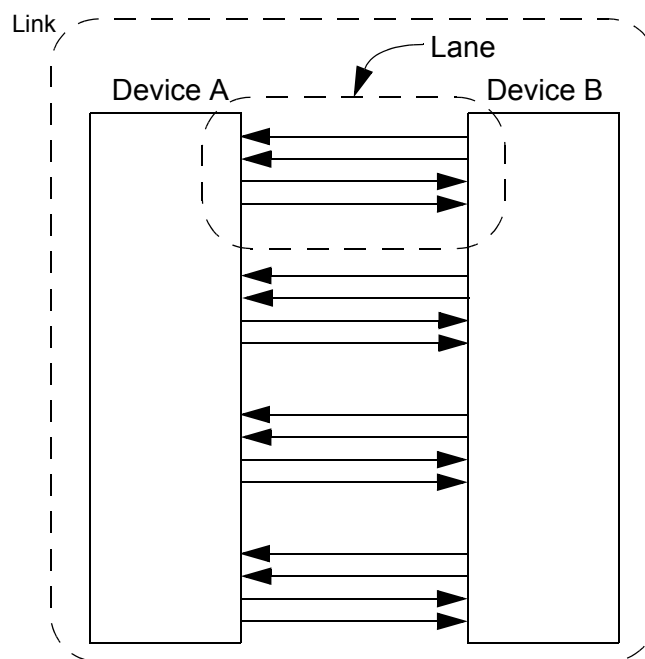


Figure 20: A PCI Express Lane

With a PCI Express lane, there is no arbitration for a shared Bus, as the Transmitting device is always directly connected with a Receiving device and each device is always both a transmitting device and a receiving device.

A PCI Express Link is defined as the connection between two devices and includes one or more lanes. The *PCI Express Specifications* define x1, x2, x4, x8, x12, x16, and x32 lanes for serial Links. The following figure illustrates a x4 (four-lane) Link.



**Figure 21: PCI Express x4 Link**

PCI Express maintains compatibility with PCI software by simulating a PCI bus across each Link.

## E.3 The PCI Express Fabric

The following figure illustrates a typical PCI Express fabric (or topology).

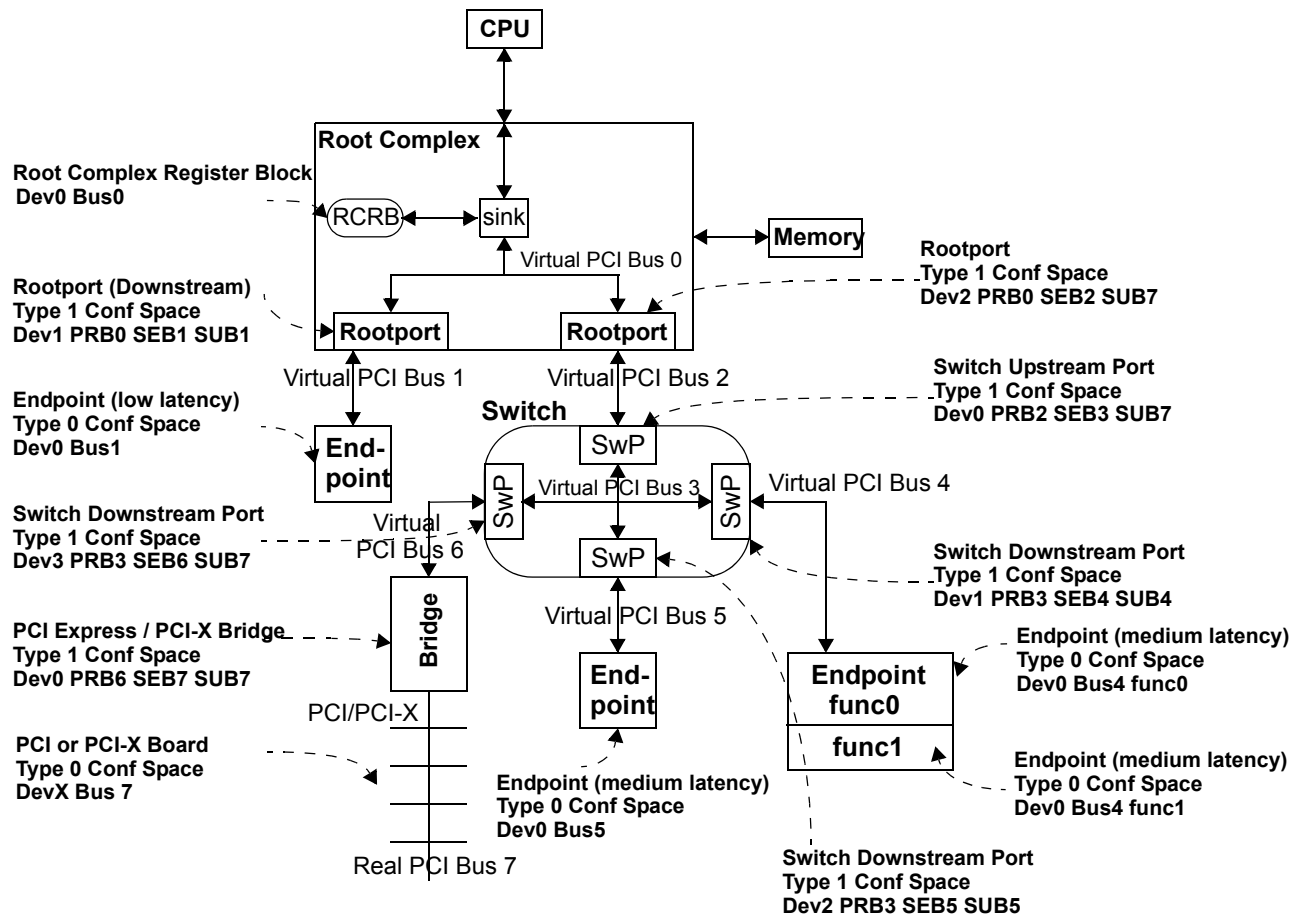


Figure 22: A Typical PCI Express Fabric

### E.3.1 Root Complex

In the example above, the Root Complex is composed of one Root Complex Register Block (RCRB) associated with the sink unit (entry to the PCI Express fabric hierarchy) and two Rootports. The sink unit and the two Rootports constitute a virtual PCI bus segment that has Bus Number 0. This means that the Root Complex implements a virtual embedded switch between the Rootports and transfers are allowed directly between them (this is an optional feature of the Root Complex).

The PCI configuration software programs each Rootport as a Type 1 Configuration space. In the example, the first Rootport is Bus 0, Device 1 and the second Bus 0, Device 2.

The sink unit represents the start of the PCI Express fabric and is equivalent to a virtual PCI host port that can generate all transaction types (Memory, I/O, Completion, and Message) and also relay Request and Completion transactions used to access the central memory.

Rootport Device 1 is connected to an Endpoint through virtual PCI Bus Segment 1 and has the following Type 1 Configuration Space parameters:

- Primary Bus Number 0
- Secondary Bus Number 1
- Subordinate Bus Number 1 (only one bus number is located downstream of the Rootport)

Rootport device 2 is connected to a PCI Express Switch and has the following configuration parameters:

- Primary Bus Number 0
- Secondary Bus Number 2
- Subordinate Bus Number 7 (bus segments 2 through 7 are located behind the Rootport)

The Endpoint component located behind Rootport Device 1 is a low latency Endpoint, for example, a graphic



board where latency is critical for optimal performance. An Endpoint device number is always 0 and its Bus number is programmed by the PCI configuration software (Bus 1, in this example).

### E.3.2 Endpoint

All PCI Express Endpoints implement a Type 0 Configuration Space and have Device number 0. In the example (Figure 22), the two Endpoints located behind the Switch have Bus Numbers 4 and 5. The Endpoint located behind Bus 4 implements two different functions (func0 and func1). The Endpoints located behind Bus Numbers 4 and 5 are considered medium latency components due to the fact that they are behind a Switch component and not directly connected to the Root Complex. In general, a PCI Express Endpoint component is considered to have high latency if two or more Switch components lie between it and the Root Complex.

The PCI Endpoint components located behind the Bridge are also implemented as Type 0 Configuration space components and respond only to Type 0 Configuration access requests (if their PCI\_IDSEL pin is asserted).

### E.3.3 Switch

In the example (Figure 22), the PCI Express Switch component implements four PCI Express ports, one upstream and three downstream. Each port is equivalent to a PCI Bridge and implements a Type 1 Configuration space. Internal routing between ports is accomplished by a virtual PCI Bus segment.

The following table describes the configuration of each Switch port in Figure 22.

**Table 71: Configuration of Switch ports**

Device Number	0 (upstream)	1 (downstream)	2 (downstream)	3 (downstream)
Primary Bus Number	2	3	3	3
Secondary Bus Number	3	4	5	6
Subordinate Bus Number	7	4	5	7

### E.3.4 Bridge

The PCI Express / PCI Bridge component implements a common Type 1 Configuration space for both sides of the bridge: the PCI Express Core and the PCI component Core. The mechanism to handle transactions between the two is described in the *PCI Express Base Specification Revision 2.0* (as well as *Revisions 1.1 and 1.0a*). In Figure 22, the Bridge has the following parameters:

- Device number 0 (mandatory)
- Primary Bus Number 6 (PCI Express Core Link)
- Secondary Bus Number 7 (PCI bus)
- Subordinate Bus Number 7 (no other PCI / PCIe Bridges are located on the PCI bus)

## E.4 Types of Transactions

The following table summarizes the different types of transactions possible in the PCI Express fabric:

**Table 72: PCI Express transaction types and characteristics**

Transaction Type	How request is handled	Routing method	Notes
Memory Write	Posted (no Completion is required)	Address	Memory Write Requests use Posted TLPs and can have data payloads of up to 4 KBytes (depending on the Max_Payload_Size parameter of the Configuration Space). Memory Write Requests can use 32-bit address formatting or 64-bit address formatting.
Memory Read	Non-Posted (Completion is required)	Address	Memory Read Requests use Non-Posted TLPs and have no payload. A dedicated register of the configuration space defines the maximum Read Request size. Please see the following section for details concerning Completion transactions. Memory Read Requests can use 32-bit address formatting or 64-bit address formatting.
I/O Write	Non-Posted	Address	I/O Request transactions use Non-Posted TLPs. I/O Write Request transactions have a data payload of 1DW. I/O Requests always require a Completion packet, whether the request is returned successful or aborted. Please see the following section for details concerning Completion transactions.
I/O Read	Non-Posted	Address	
Configuration Write	Non-Posted	ID	Configuration Requests use Non-Posted TLPs. Configuration Write requests have data payloads of 1 DW. Each Configuration request requires a Completion regardless of whether the request was accepted. Please see the following section for details concerning Completion transactions.
Configuration Read	Non-Posted	ID	
Message	Posted	Address	In addition to the three PCI transactions (Memory, I/O, and Configuration), PCI Express introduces a fourth type of transaction for Messages. Message transactions support power management requests and emulate PCI legacy virtual pins (such as INT, PME, PERR, and SERR).

A Completion Transaction is a distinct package generated in the PCI Express fabric in response to a Read or Write Request. Like Configuration Requests, Completion transactions are routed by Transaction ID.

Completion transactions vary in nature depending on the reason for their generation. The following table

describes differences in Completion transactions.

**Table 73: Completion transaction characteristics**

Completion transactions generated in response to a...	have these characteristics
Memory Request	<p>Completion transactions are generated for Memory Read Requests but not Memory Write requests, since no completion is required for the latter.</p> <p>Completions can have data payloads of up to 4 KBytes, depending on the Maximum Payload Size. Note that large Read Request transactions (of 4 KBytes, for example) might be divided into several smaller Completion Packets (64, 128, or 256 bytes on a 64 or 128 address boundary) in order to reduce overall latency and to optimize data flow within each Switch. Completion transactions may only be divided if they have an end address corresponding to the Read Completion Boundary (RCB) parameter, which is 64 or 128 bytes for Endpoints, Root Complexes, and Bridges, and 64 bytes for Switches.</p> <p>Note that the Core does not check for violations of the Read Completion Boundary (RCB).</p>
I/O Request	<p>Completion transactions are generated for both Read and Write I/O requests.</p> <p>Completion packets generated in response to I/O Write requests have no data payload, whereas a completion packet in response to an I/O Read request has a data payload of 1 DW. I/O Requests always require a Completion transaction regardless of return status (accepted, successful, unsupported, aborted, or retry).</p>
Configuration Request	<p>Completion transactions are generated for both Read and Write Configuration requests.</p> <p>Completion packets have data payloads of 1 DW if the request is accepted. Configuration Requests always require a Completion transaction regardless of return status (accepted, successful, unsupported, aborted, or retry).</p>

## E.5 Routing Rules

The PCI Express fabric uses the following routing rules:

- All Memory and I/O transactions are routed by address (BAR decoding for Endpoint Type 0 configuration spaces, and Memory Mapped I/O for Rootport, Switch, and Bridge Type 1 Configuration spaces).
- Configuration, Message, and Completion transactions are routed by the Transaction ID of the destination component (Bus, Device, and Function number) and comply with Type 1 Configuration Space routing specifications.

Message transactions are independent of the PCI configuration software and have different types of routing. See [Section E.5.3](#) for details.

The TLP header includes information about the type of the transaction and whether the transaction is routed by address or ID.

## E.5.1 Routing by Address

Transactions that are routed by address compare the destination address (included in the TLP header) with the registers of the component.

**Table 74: Routing by Address Rules**

Component Type	Routing
Type 0 Components (Endpoint)	<p>If the address of the transmitted TLP lies within one of the component's implemented Base Address Registers (BARs), the component will accept the TLP.</p> <p>If the address of the TLP does not lie within the component's BARs, the TLP is rejected and an error is generated.</p>
Type 1 Components (Rootport, Switch, or Bridge)	<p>If a device receives a request on its Primary Bus and the destination address is located within one of the defined address windows (as defined by its registers), the request is forwarded to the Secondary Bus.</p> <p>If a device receives a request on its Secondary Bus and the destination address is located outside of the defined address window, the request is forwarded to the Primary Bus.</p>

## E.5.2 Routing by ID

Regardless of the type of PCI Express component, when a Type 1 Configuration Space port receives a TLP on its Primary Bus, the following rules apply:

- If the selected Bus Number is equal to its Secondary Bus number, the Type 1 Configuration request is transformed into a Type 0 Configuration request according to the standard PCI rules.
- In all other cases, the Type 1 Configuration request remains a Type 1 Configuration request and is transferred to the Secondary Bus (internal or external).
- If the selected Bus Number does not fall between the Secondary Bus number and the Subordinate Bus number, the transaction is discarded.

## E.5.3 Implicit Routing

Messages do not use Address or ID routing, but rely on other mechanisms referred to as "implicit" because the destination is implied by the routing type (see section 2.2.8 of the *PCI Express Base Specification Revision 2.0, 1.1 or 1.0a* for more information). Implicit routing depends upon the inherent knowledge a PCI Express component has relating to upstream and downstream transmissions within the fabric.

"Implicit" Routing types include the following:

- Routed to Root Complex
- Broadcast from Root Complex
- Local (Terminate at Receiver)
- Gathered and routed to Root Complex
- reserved (Terminate at Receiver)

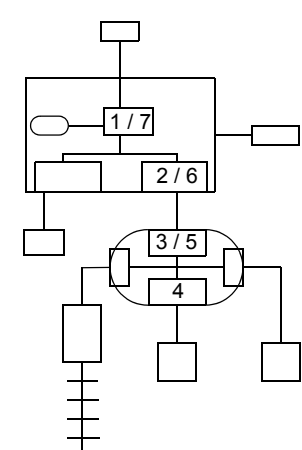
## E.5.4 Routing Examples

The following section outlines two hypothetical transactions that correspond to [Figure 22](#).

### E.5.4.1 Configuration Write Transaction

Configuration transactions use ID routing (routing based on the Bus, Device, and Function numbers of the destination component).

**Table 75: Configuration Write transaction steps**

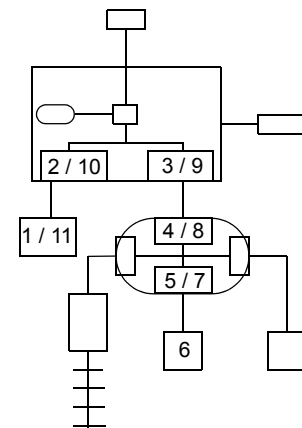
Step	Component	ID	Description	 <p><b>Figure 23: Tracing a Write Transaction through the fabric</b></p>
1	Sink Unit / CPU	Dev0 Bus0	The CPU instructs the sink unit to generate a Configuration Write transaction directed to Bus Number 3, Device 2.	
2	Rootport	Dev2 PRB0 SEB2 SUB7	The Rootport receives the transaction and determines that the destination Bus (3) is located within its Bus Address window (the range defined by its Secondary Bus number and Subordinate Bus number, in this case 2-7). It accepts the transaction in order to forward it to its Secondary Bus. The target Bus (3) is not equal to the Secondary Bus (2), and the transaction remains a Configuration Type 1 transaction.	
3	Switch Upstream Port	Dev0 PRB2 SEB3 SUB7	The Switch's Upstream Port receives the transaction and determines that the destination Bus (3) is located within its Bus Address window and accepts the transaction in order to forward it to its Secondary Bus (the Switch's internal virtual PCI bus). The target Bus (3) is equal to the Secondary Bus (3), and the Type 1 Configuration transaction is transformed into a Type 0 Configuration transaction.	
4	Switch Downstream Port	Dev2 PRB3 SEB5 SUB5	Each downstream port in the Switch receives the Type 0 Configuration transaction on virtual Bus 3 and decodes the Device number (2). Port Device 2 recognizes itself as the Completer, accepts the transaction, and generates a Completion TLP using a Transaction ID (Bus 0, Device 0).	
5	Switch Upstream Port	Dev0 PRB2 SEB3 SUB7	The Completion TLP is accepted by the Secondary Bus of the Switch's upstream port, which determines that the target Bus Number (0) does not lie within its Bus Address window, and the transaction is forwarded to its Primary Bus.	
6	Rootport	Dev2 PRB0 SEB2 SUB7	Likewise, the Rootport forwards the transaction back to Virtual Bus 0.	
7	Sink Unit / CPU	Dev0 Bus0	Finally, the sink unit accepts the packet and passes it back to the CPU.	

### E.5.4.2 Memory Read Transaction

Steps 1-6 of the following example describe a Memory Read request, which uses address routing (routing based on either a 64-bit format associated with 4 DW headers or a 32-bit format associated with 3 DW headers). For this example, it is assumed that address windows are correctly defined for each Type 1 Configuration Space component and that the Completer Endpoint Base Address Register (BAR) is correctly defined.

Steps 7-14 describe routing of the Completion transaction generated in response to the Memory Read request. As in the first example (Configuration transaction), Completion transactions use ID routing (routing based on the Bus, Device, and Function numbers of the destination component).

**Table 76: Memory Read transaction steps**

Step	Component	ID	Description	 <p><b>Figure 24: Tracing a Read Transaction through the fabric</b></p>
1	Endpoint	Dev0 Bus1	The Endpoint component located behind Rootport Device 1 generates a Memory Read request directed to the Endpoint located behind the Switch's Port device 2.	
2	Rootport	Dev1 PRB0 SEB1 SUB2	The Secondary Bus of the Rootport (Device 1) receives the transactions, determines that the address lies outside of its address window, and forwards the transaction through its Primary Bus to the Virtual PCI Bus 0.	
3	Rootport	Dev2 PRB0 SEB2 SUB7	The Primary Bus of the Rootport (Device 2) determines that the destination address lies within its prefetchable address window and forwards the transactions through its Secondary Bus to the Switch.	
4	Switch Upstream Port	Dev0 PRB2 SEB3 SUB7	The Primary Bus of the upstream port of the Switch determines that the destination address lies within its address window and forwards the transactions through its Secondary Bus to the virtual PCI Bus 3.	
5	Switch Downstream Port	Dev2 PRB3 SEB5 SUB5	The Primary Bus of downstream port Device 2 of the Switch determines that the destination address lies within its address window and forwards the transactions through its Secondary Bus to the virtual PCI Bus 5.	
6	Endpoint	Dev0 PRB5	The Endpoint (the Completer, in this example) receives the transaction and determines that the address corresponds to one of its BARs and accepts the request. The Endpoint responds to the request and generates one or more Completion transactions to send data back to the requester.	
Steps 7-14 describe routing of the Completion transaction generated in response to the Memory Read request.				
7	Switch Downstream Port	Dev2 PRB3 SEB5 SUB5	Completion transactions use ID routing, and in this example target the requesting component (Endpoint Bus1 Dev0). The Secondary Bus of the downstream port Device 2 determines that the destination Bus (1) lies outside of its Bus window (5-5) and forwards the transaction through its Primary Bus to the virtual PCI Bus 3.	
8	Switch Upstream Port	Dev0 PRB2 SEB3 SUB7	The Secondary Bus of the upstream port determines that the destination Bus (1) lies outside of its Bus window (3-7) and forwards the transaction through its Primary Bus to the Rootport.	
9	Rootport	Dev2 PRB0 SEB2 SUB7	The Secondary Bus of the Rootport determines that the destination Bus (1) lies outside of its Bus window (2-7) and forwards the transaction through its Primary Bus to the virtual PCI bus 0.	
10	Rootport	Dev1 PRB0 SEB1 SUB2	The Primary Bus of the Rootport determines that the destination Bus (1) lies within its Bus window (1-1) and forwards the transaction through its Secondary Bus to the Endpoint.	
11	Endpoint	Dev0 Bus1	The Endpoint recognizes itself as the destination component, verifies the tag of the transaction, and processes the transmitted information.	

## E.6 Flow Control

### E.6.1 Traffic Classes (TCs)

TCs allow for differentiated services, permitting you to prioritize the flow of certain data through the fabric. A Link must implement at least one TC (TC0) and can implement up to eight TCs, depending on your design.

### E.6.2 Virtual Channels (VCs)

VCs allow for multiple independent paths of data flow over a single Link. A Link must implement at least one VC (VC0) and can implement up to eight VCs, according to your design.

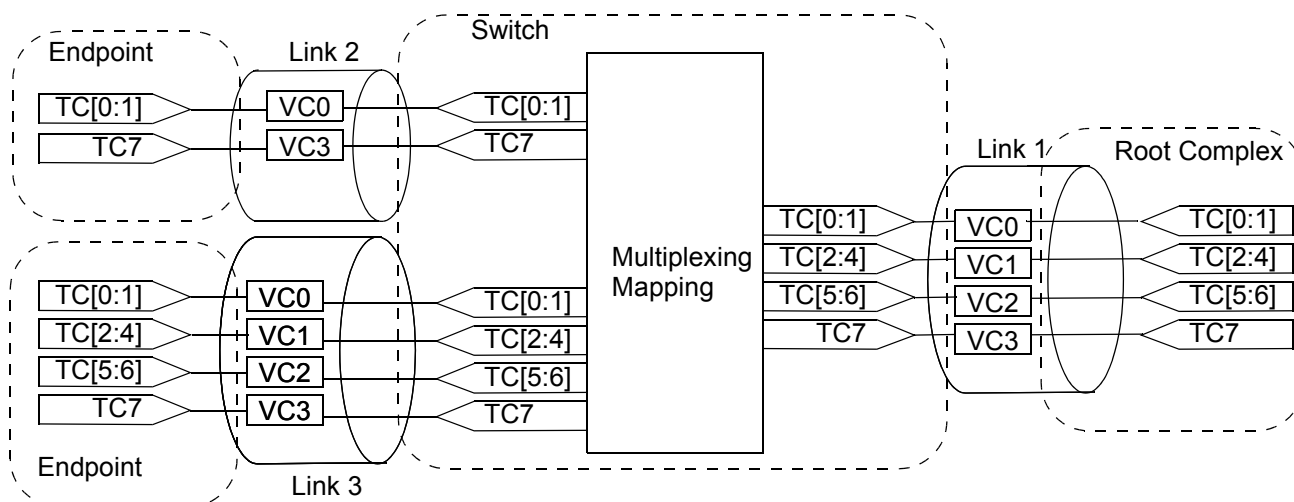
The number of VCs initialized across a Link has no relation to the number of lanes implemented by the Link. For example, a x1 component could have eight VCs, and a x4 component could have one VC.

If two components on either side of a Link implement a different number of VCs, only the number of VCs the two components have in common are initialized (see [Figure 26](#)).

### E.6.3 TCs and VCs

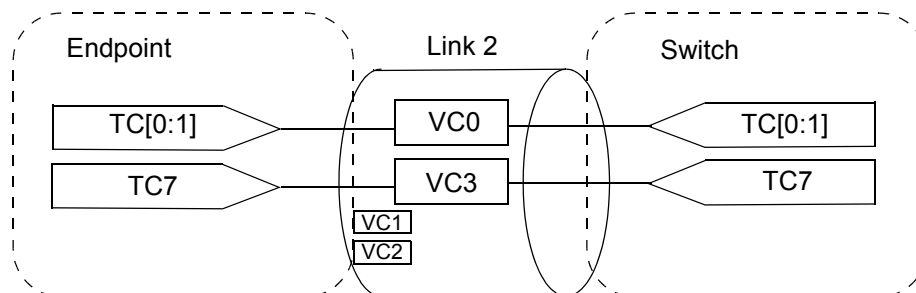
The concept of TCs associated with VCs facilitate data flow in the fabric, allowing you to determine what percentage of a particular Link should be devoted to a particular kind of data transfer. This, in turn, helps to avoid congestion and permits isochronous traffic.

The following figure illustrates the function of Virtual Channels and Traffic Classes in the fabric.



**Figure 25: Flow Control through Virtual Channels (VCs) and Traffic Classes (TCs)**

Note that the figure above does not specify how many VCs are implemented per component. Consider Link 2, enlarged in the following figure:

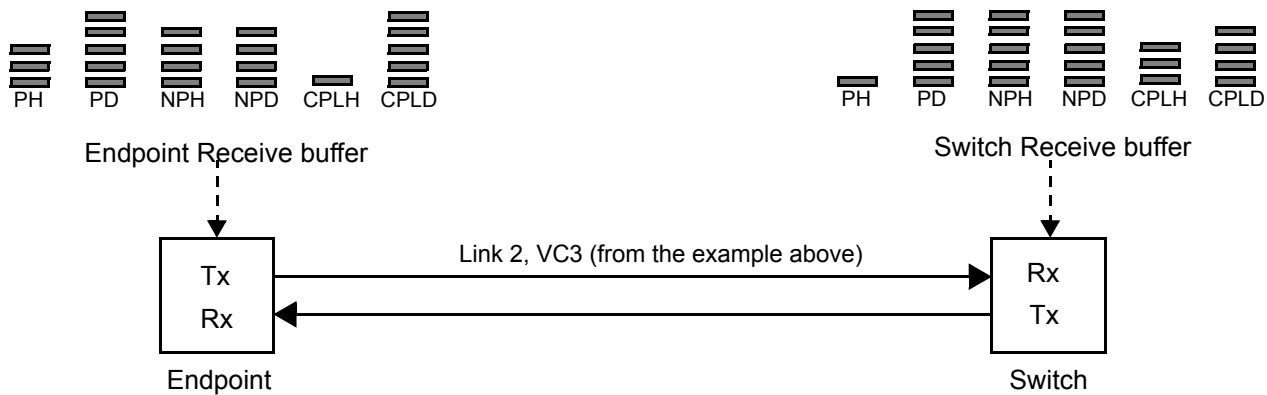


**Figure 26: Flow Control through a single Link**

The Endpoint component might also have VC1 and VC2 implemented, but the Link only initializes the number of common VCs shared across a Link.

## E.6.4 Receive Buffer

Each component commits a certain number of resources (determined by the user) to the Receive buffer. The Receive buffer is located in the Transaction Layer and accepts incoming TLPs from the Link and then sends them to the Application Layer for processing. Receive buffer resources are either implemented per VC or per Link. The following figure enlarges VC3 from Figure 25 and illustrates the various buffers that, taken together, make up the Receive buffer.



**Figure 27: Receive buffers for a Virtual Channel**

The Receive buffer stores TLPs based on the type of a transaction, not the TC of a transaction. Types of transactions include Posted transactions, Non-Posted transactions, and Completion transactions.

A transaction always has a header but does not necessarily have data. The Receive buffer accounts for this distinction, maintaining separate resources for the header and data of each type of transaction. To summarize, distinct buffer resources are maintained per initialized VC for each of the following elements:

- Posted transactions, header (PH)
- Posted transactions, data (PD)
- Non-Posted transactions, header (NPH)
- Non-Posted transactions, data (NPD)
- Completion transactions, header (CPLH)
- Completion transactions, data (CPLD)

Note that the Receive buffer levels on one side of a Link have no relation to the Receive buffer levels on the other side of a Link.

The size of the Receive buffer has a significant impact on system performance. The smallest possible size is the Maximum Payload Size (Max\_Payload\_Size), but the Receive buffer is typically set to at least four times the Maximum Payload Size for the following reasons:

- so that it can store a maximum-sized TLP in the buffer and forward a second TLP to the Application Layer
- so that it can handle multiple Completions of minimum payload size. For example, a component set to handle 16 outstanding requests, each of which might have four corresponding Completion packets, would need a minimum Receive buffer size of  $16 \times 4 = 64$  DW.

## E.6.5 Flow Control Credits

A component advertises Buffer space availability with Flow Control credits. FC credits are maintained for each of the six Receive buffers and are transferred using FC Packets, a type of Data Link Layer Packet (DLLP).

The transmitting side of a Link will not send a transaction if the receiving side hasn't advertised enough FC credits (header and data credits) for that particular type of transaction.



The following table offers an example of advertised FC credits for an Endpoint component.

**Table 77: Example of an Endpoint's advertised credits at and after Link initialization and the effect on Flow Control**

Type of FC credit	Advertised Credits at Link initialization	Advertised Credits at Link initialization + n clock cycles	Transmission permitted...
Posted Header	16	0	No: Sufficient credits for both header and data must be advertised before a packet is transmitted.
Posted Data	128	96	
Non-Posted Header	16	16	Yes
Non-Posted Data	16	16	
Completion Header	infinite	infinite	Yes: Please see the following section, Deadlock avoidance, for an explanation of infinite credits.
Completion data	infinite	infinite	

The unit of a single FC credit differs between header and data:

- Header (maximum-sized header + digest<sup>1</sup>)
  - 4 DWs for Completion transactions
  - 5 DWs for Request transactions
- Data: 4 DWs (16-bytes aligned)

FC Credits are initialized for each VC with maximum credits and then updated periodically as TLPs are extracted from the Receive buffer of the receiving side of a Link. Note that one DLLP can update FC Credits associated with one or more TLPs.

## E.6.6 Deadlock Avoidance

As in all Switch and Bridge architectures, deadlock occurs when a component can only proceed when one of its internal resources is freed. PCI Express prevents deadlock with two complementary mechanisms:

- **Reordering of TLPs:** Component resources are freed when Posted Write requests are executed or when a component receives the final Completion TLP in response to a Read request. More specifically, Non-Posted requests must give priority to Posted and Completion transactions.
- **Infinite credits:** Endpoints and Rootports must advertise infinite Completion credits in order to prevent deadlock. To do so, they are not allowed to initiate Read Request transactions if they do not have sufficient Completion buffer space in order to store the maximum number of Completion transactions that might be generated in response to the Read Request.

Note: Other reordering rules govern interactions between Posted requests, Non-Posted requests, and Completion transactions depending on the Traffic Class (TC) and on the “relaxed ordering” bit of the TLP. These additional reordering rules have no effect on deadlock avoidance but can facilitate the flow of global traffic within a fabric.

1. The TLP digest is the end-to-end CRC located just before the LCRC at the end of the TLP. It is signaled with the TD bit set to 1 in the header.

## E.7 Error Handling

### E.7.1 Error Recovery

The Data Link Layer handles error recovery. It is based on 32-bit Cyclic Redundant Check (CRC) error detection, TLP sequence number, Replay buffer, and ACK/NAK DLLP exchange. A typical error check might transpire as follows:

- Step 1 Transmitter: On the transmit side of the Link, the Data Link Layer adds a sequence number to the TLP and generates a 32-bit CRC in order to protect the complete packet. The transmitter also stores the sent packet in its Replay buffer.
- Step 2 Receiver: On the receive side of the Link, the Data Link Layer verifies the CRC and the sequence number and generates an Acknowledgement / Negative Acknowledgement (ACK/NAK) DLLP in order to report to the transmitter if the packet has been correctly received.
- Step 3 Transmitter: If the received ACK/NAK DLLP is negative (NAK), the transmitter must end its current transmission and re-send the TLP that caused the NAK response. If, on a second try, the packet is successfully transmitted (ACK DLLP), the TLP is purged from the Replay buffer. If the transmitter has not received an ACK reply after three retries, the Link is directed to recovery, that is retraining of the Link.

Note: DLLPs are also protected by a 16-bit CRC, but no acknowledge or error recovery mechanism exists for this check. Instead, DLLPs are periodically generated in order to ensure that the proper information is transmitted.

### E.7.2 End-to-End Cyclic Redundancy Check (ECRC)

PCI Express maintains data integrity across two Endpoints of the fabric with the TLP digest (also called ECRC). The TD bit (TLP Digest bit) of the TLP header indicates whether the TLP includes a TLP digest.

Switch devices change packet sequence numbers between ingress and egress ports and can introduce errors as they recalculate the CRC. The TLP digest is an optional feature used in high reliability systems to check for introduced errors.

### E.7.3 Replay buffer

The Replay buffer, located in the Data Link Layer and common to all VCs, stores a copy of a transmitted TLP until the transmitted packet is acknowledged by the receiving side of the Link. Each stored TLP includes the Header, an optional data payload (of which the maximum size is determined by the Maximum Payload Size parameter), an optional ECRC, the sequence number, and the LCRC field.

The receiving side of the Link acknowledges reception of a TLP with transmission to the transmitting side of an ACK DLLP. In the case of CRC error on the receiving side, a NAK DLLP is sent to the transmitting side, which retrieves the TLP from the Replay buffer and sends it again.

The user is responsible for setting the size of the Replay buffer, which should be of a sufficient size so that TLP transmission is not delayed because of a full buffer. Replay buffer resources are only freed upon reception of an ACK DLLP, which means that Link latency (associated with DLLP transmission and/or implementation of the Physical Layer) affects the ideal size of the Replay buffer. In general, the Replay buffer size should be at least twice the maximum TLP size.

### E.7.4 Completion Timeout

Endpoint and Rootport components use a timeout mechanism (which is design-specific) for failed Read Request transactions in order to report errors to the Rootport (using Error Messages) and to free Completion resources.