Arria V Hard IP for PCI Express

User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

UG-01105-1.0

Document last updated for Altera Complete Design Suite version: 11.1
Document publication date: November 2011

Feedback Subscribe

Contents

1. Datasheet

This document describes the Altera® Arria® V Hard IP for PCI Express®. PCI Express is a high-performance interconnect protocol for use in a variety of applications including network adapters, storage area networks, embedded controllers, graphic accelerator boards, and audio-video products. The PCI Express protocol is software backwards-compatible with the earlier PCI and PCI-X protocols, but is significantly different from its predecessors. It is a packet-based, serial, point-to-point interconnect between two devices. The performance is scalable based on the number of lanes and the generation that is implemented. Altera offers a configurable hard IP block in Arria V devices for both Endpoints and Root Ports that complies with the *PCI Express Base Specification 2.1*. Using a configurable hard IP block, rather than programmable logic, saves significant FPGA resources. The hard IP block is available in ×1, ×4, and ×8 configurations. The Table 1–1 shows the aggregate bandwidth of a PCI Express link for the available configurations. The protocol specifies 2.5 giga-transfers per second for Gen1 and 5 giga-transfers per second for Gen2. Table 1–1 provides bandwidths for a single transmit (TX) or receive (RX) channel, so that the numbers double for duplex operation. Because the PCI Express protocol uses 8B/10B encoding, there is a 20% overhead which is included in the figures in Table 1–1

Table 1–1. PCI Express Throughput

	Link Width		
	×1	×4	×8
PCI Express Gen1 Gbps (1.x compliant)	2.5	10	20
PCI Express Gen2 Gbps (2.1 compliant)	5	20	—

Refer to the *PCI Express High Performance Reference Design* for more information about calculating bandwidth for the hard IP implementation of PCI Express in many Altera FPGAs.

Features

Altera's Arria V Hard IP for PCI Express IP supports the following key features:

- Complete protocol stack including the Transaction, Data Link, and Physical Layers is hardened in the device.
- Multi-function support for up to eight Endpoint functions.
- Support for ×1, ×4, and ×8 configurations.
- Optional end-to-end cyclic redundancy code (ECRC) generation and checking and advanced error reporting (AER) for high reliability applications.
- Qsys support using the Avalon® Streaming (Avalon-ST) with a 64- or 128-bit interface to the Application Layer.
- Extended credit allocation settings to better optimize the RX buffer space based on application type.

- Qsys walkthough demonstrating parameterization, design modules and connectivity.
- Easy to use:
  - Easy parameterization.
  - Substantial on-chip resource savings and guaranteed timing closure.
  - Easy adoption with no license requirement.

Table 1–2 summarizes the IP core's features.

Table 1–2. Hard IP for PCI Express Features

Feature	Avalon-ST Interface
MegaCore License	Free
Endpoint	Supported
Legacy Endpoint (1)	Supported
Root port	Supported
Gen1 (2)	×1, ×4, ×8
Gen2 (2)	×1, ×4
Multiple functions	Supports up to 8 functions for Endpoints and Legacy Endpoints
MegaWizard Plug-In Manager design flow	Supported
Qsys design flow	Supported
64-bit Application Layer interface	Supported
128-bit Application Layer interface	Supported
Transaction layer packet type (TLP) (3)	All
Payload size	128–512 bytes
Number of tags supported for non-posted requests	32 or 64
Low power mode using 62.5 MHz clock	Supported
ECRC forwarding on RX and TX	Supported
Number of MSI requests	16
MSI-X	Supported
Legacy interrupts	Supported
Expansion ROM	Supported

Notes to Table 1–2:

(1) Not recommended for new designs.

(2) ×2 is supported by down training from ×4 or ×8 lanes.

(3) Refer to Appendix A, Transaction Layer Packet (TLP) Header Formats for the layout of TLP headers.

The purpose of the *Arria V Hard IP for PCI Express User Guide* is to explain how to use the Arria V Hard IP for PCI Express and not to explain the PCI Express protocol. Although there is inevitable overlap between these two purposes, this document should be used in conjunction with an understanding of the following PCI Express specifications: *PHY Interface for the PCI Express Architecture PCI Express 2.0* and *PCI Express Base Specification 2.1*.

Release Information

Table 1–3 provides information about this release of the PCI Express Compiler.

Table 1–3. PCI Express Compiler Release Information

Item	Description
Version	11.1
Release Date	November 2011
Ordering Codes	No ordering code is required
Product IDs	There are no encrypted files for the Arria V Hard IP for PCI Express. The Product ID and Vendor ID are not required because this IP core does not require a license.
Vendor ID	

Altera verifies that the current version of the Quartus® II software compiles the previous version of each IP core. Any exceptions to this verification are reported in the *MegaCore IP Library Release Notes and Errata*. Altera does not verify compilation with IP core versions older than one release.

Device Family Support

Table 1–4 shows the level of support offered by the Arria V Hard IP for PCI Express.

Table 1–4. Device Family Support

Device Family	Support
Arria V	Preliminary. The IP core is verified with preliminary timing models. The IP core meets all functional requirements, but is still undergoing characterization. It can be used in production designs with caution.
Other device families	Refer to the following user guides for other device families: ■ *IP Compiler for PCI Express User Guide* ■ *Stratix V Hard IP for PCI Express User Guide* ■ *Cyclone V Hard IP for PCI Express User Guide*

Configurations

The Arria V Hard IP for PCI Express includes a full hard IP implementation of the PCI Express stack including the following layers:

- Physical (PHY)
- Physical Media Attachment (PMA)
- Physical Coding Sublayer (PCS)
- Media Access Control (MAC)
- Data Link Layer (DLL)
- Transaction Layer (TL)

Optimized for Altera devices, the Arria V Hard IP for PCI Express supports all memory, I/O, configuration, and message transactions. It has a highly optimized Application Layer interface to achieve maximum effective throughput. You can customize the Hard IP to meet your design requirements using either the MegaWizard Plug-In Manager or the Qsys design flow.

Figure 1–1 shows a PCI Express link between two Arria V FPGAs. One is configured as a Root Port and the other as an Endpoint.

Figure 1–1. PCI Express Application with a Single Root Port and Endpoint

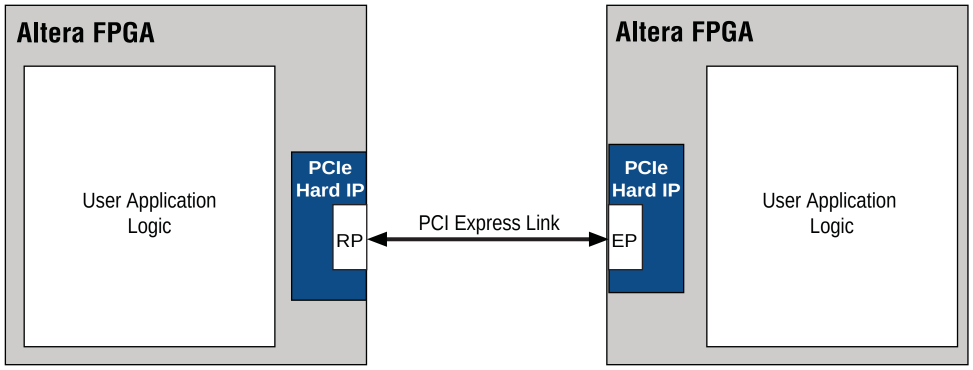
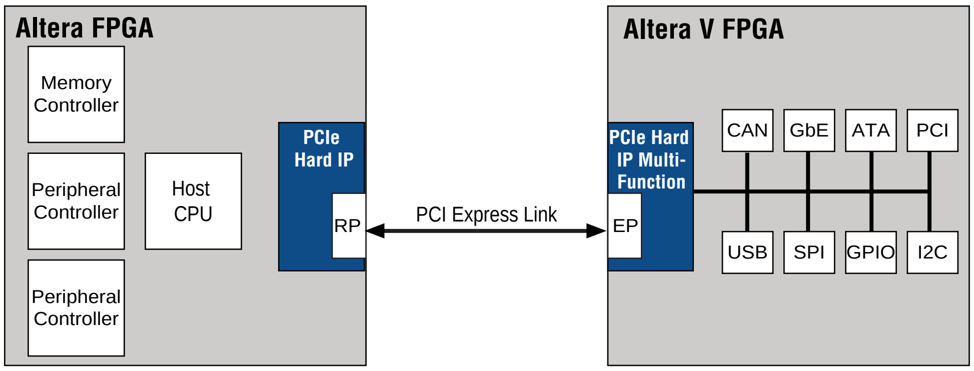


Figure 1–2 shows a PCI Express link between two Altera FPGAs. One is configured as a Root Port and the other as a multi-function Endpoint. The FPGA serves as a custom I/O hub for the host CPU. In the Arria V FPGA, each peripheral is treated as a function with its own set of Configuration Space registers. Eight multiplexed functions operate using a single PCI Express link.

Figure 1–2. PCI Express Application with an Endpoint Using the Multi-Function Capability



Debug Features

The Arria V Hard IP for PCI Express includes debug features that allow observation and control of the Hard IP for faster debugging of system-level problems. For more information about debugging refer to Chapter 15, Debugging.

IP Core Verification

To ensure compliance with the PCI Express specification, Altera performs extensive validation of the Arria V Hard IP Core for PCI Express.

Altera's simulation environment uses multiple testbenches that consist of industry-standard BFMs driving the PCI Express link interface. A custom BFM connects to the application-side interface.

Altera performs the following tests in the simulation environment:

- Directed and pseudo random stimuli are applied to test the Application Layer interface, Configuration space, and all types and sizes of TLPs.
- Error injection tests that inject errors in the link, TLPs, and Data Link Layer Packets (DLLPs), and check for the proper responses
- PCI-SIG® Compliance Checklist tests that specifically test the items in the checklist
- Random tests that test a wide range of traffic patterns

Performance and Resource Utilization

Because the IP core is implemented in hardened logic, it uses less than 1% of Arria V resources.

Depending on the speed of the variant, soft calibration logic may be required, with more logic required for more lanes. The amount of additional logic for calibration for the transceiver modules is pending characterization of the Arria V device.

Recommended Speed Grades

Table 1–5 lists the recommended speed grades for the supported link widths and Application Layer clock frequencies. The speed grades listed are the only speed grades that close timing. Altera recommends setting the Quartus II Analysis & Synthesis Settings **Optimization Technique** to **Speed**.

Refer to *"Setting Up and Running Analysis and Synthesis* in Quartus II Help for information about optimizing synthesis.

Refer to *Area and Timing Optimization* in volume 2 of the *Quartus II Handbook* for more information about how to effect the **Optimization Technique** setting.

Table 1–5. Device Family Link Width Application Frequency Recommended Speed Grades (Part 1 of 2)

Link Speed	Link Width	Application Clock Frequency (MHz)	Recommended Speed Grades
Gen1–2.5 Mbps	×1	125	–4, –5, –6
	×4	125	–4, –5, –6
	×8	125	–4, –5, –6 (2)

Table 1–5. Device Family Link Width Application Frequency Recommended Speed Grades (Part 2 of 2)

Link Speed	Link Width	Application Clock Frequency (MHz)	Recommended Speed Grades
Gen2–5.0 Mbps	×1	62.5 (1)	–4, –5 (2)
	×1	125	–4, –5 (2)
	×4	125	–4, –5 (2)

Notes to Table 1–5:

(1) This is a power-saving mode of operation.

(2) Final results pending characterization by Altera for speed grades -2, -3, and -4. Refer to the **.fit.rpt** file generated by the Quartus II software.

For details on installation, refer to the *Altera Software Installation and Licensing Manual*.

2. Getting Started

This section provides step-by-step instructions to help you quickly customize, simulate, and compile the Arria V Hard IP for PCI Express using either the MegaWizard Plug-In Manager or Qsys design flow. When you install the Quartus II software you also install the IP Library. This installation includes the following example designs for the Arria V Hard IP for PCI Express:

- Gen1 ×4 Endpoint with a 64-bit Avalon-ST interface to the Application Layer
- Gen1 ×8 Endpoint with a 128-bit Avalon-ST interface to the Application Layer
- Gen1 ×4 Root Port with a 64-bit Avalon-ST interface to the Application Layer
- Gen1 ×8 Root Port with a 128-bit Avalon-ST interface to the Application Layer

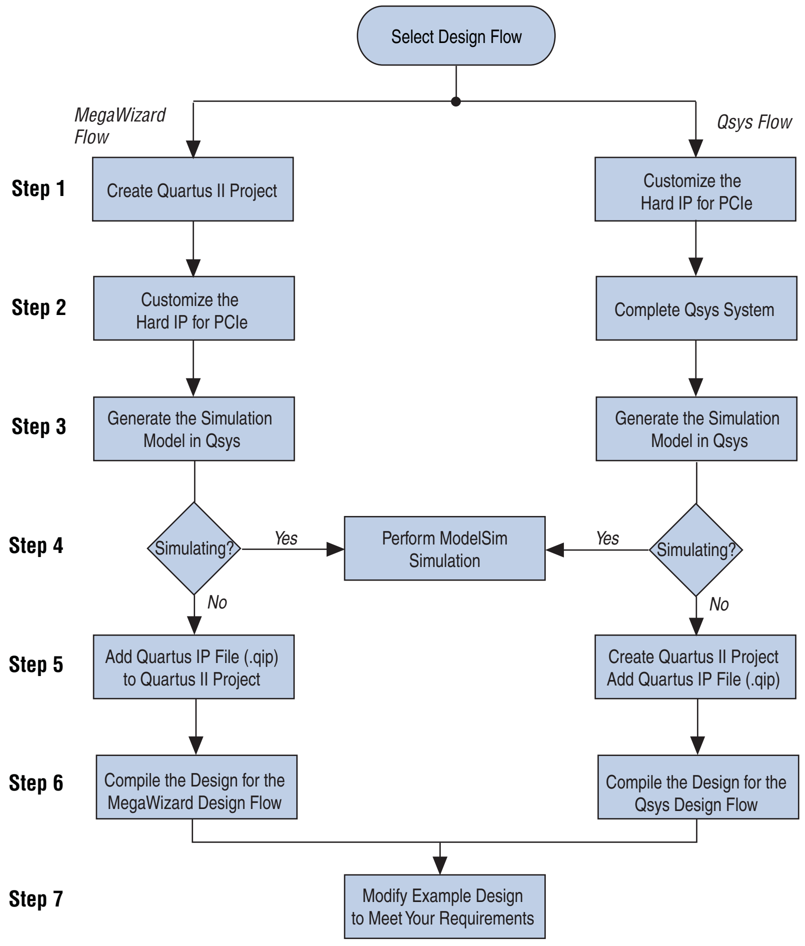
After you install the Quartus II software for 11.1, you can copy these example designs from the <*install_dir*>**/ip/altera/altera_pcie/altera_pcie_hip_ast_ed/example_design/av** directory. This walkthrough uses the Gen1 ×4 Endpoint. The Arria V Hard IP for PCI Express offers exactly the same feature set in both the MegaWizard and Qsys design flows. Consequently, your choice of design flow depends on whether you want to integrate the Arria V Hard IP for PCI Express using RTL instantiation or using Qsys, which is a system integration tool available in the Quartus II software.

For more information about Qsys, refer to *System Design with Qsys* in the *Quartus II Handbook*.

For more information about the Qsys GUI, refer to *About Qsys* in Quartus II Help.

Figure 2–1 illustrates the steps necessary to customize the Arria V Hard IP for PCI Express and run the example design.

Figure 2–1. MegaWizard Plug-In Manager and Qsys Design Flows


Select Design Flow
MegaWizard Flow
Qsys Flow
Step 1
Create Quartus II Project
Customize the Hard IP for PCIe
Step 2
Customize the Hard IP for PCIe
Complete Qsys System
Step 3
Generate the Simulation Model in Qsys
Generate the Simulation Model in Qsys
Step 4
Simulating?
Yes
Perform ModelSim Simulation
Yes
Simulating?
No
No
Step 5
Add Quartus IP File (.qip) to Quartus II Project
Create Quartus II Project Add Quartus IP File (.qip)
Step 6
Compile the Design for the MegaWizard Design Flow
Compile the Design for the Qsys Design Flow
Step 7
Modify Example Design to Meet Your Requirements

The following sections provide step-by-step instructions for both design flows. Steps 1 to 3 are different for each design flow and are described separately. Step 4 is identical for both flows and is described once. You can also skip Step 4 and proceed directly to Quartus II compilation. Step 5 and 6 are different for the two design flows and are described separately. Step 7 is the same for both flows and is described once.

You can begin by selecting one of these two design flows:

- MegaWizard Plug-In Manager Design Flow
- Qsys Design Flow

MegaWizard Plug-In Manager Design Flow

This section guides you through the steps necessary to customize the Arria V Hard IP for PCI Express and run the example testbench, starting with the creation of a Quartus II project. It includes the following steps:

- Creating a Quartus II Project
- Customizing the Endpoint in the MegaWizard Plug-In Manager Design Flow
- Understanding the Files Generated
- Generating the Simulation Model Using Qsys
- Compiling the Design in the MegaWizard Plug-In Manager Design Flow
- Modifying the Example Design

Creating a Quartus II Project

Follow these steps to copy the example design files, and create a Quartus II project.

1. Choose **Programs > Altera > Quartus II** <*version*> (Windows Start menu) to run the Quartus II software.
2. On the Quartus II File menu, click **New,** then **New Quartus II Project**, then **OK**.
3. Click **Next** in the **New Project Wizard: Introduction** (The introduction does not display if you previously turned it off.)
4. On the **Directory, Name, Top-Level Entity** page, enter the following information:
   a. The working directory for your project. This design example uses <*working_dir*>**/example_design**
   b. The name of the project. This design example uses **pcie_de_gen1_x4_ast64**

   The Quartus II software specifies a top-level design entity that has the same name as the project automatically. Do not change this name.

5. Click **Next** to display the **Add Files** page.
6. Click **Yes**, if prompted, to create a new directory.
7. Click **Next** to display the **Family & Device Settings** page.
8. On the **Device** page, choose the following target device family and options:
   a. In the **Family** list, select **Arria V**.
   b. In the **Devices** list, select **Arria V GX Extended Features**.
   c. In the **Available devices** list, select **5AGXFB3H6FF35C6ES**.

   This part number is for an engineering sample (ES) which is not capable of running Gen2 ×8 designs.

9. Click **Next** to close this page and display the **EDA Tool Settings** page.
10. From the **Simulation** list, select **ModelSim®**. From the **Format** list, select the HDL language you intend to use for simulation.

11. Click **Next** to display the **Summary** page.
12. Check the **Summary** page to ensure that you have entered all the information correctly.
13. Click **Finish** to create the Quartus II project.

Customizing the Endpoint in the MegaWizard Plug-In Manager Design Flow

This section guides you through the process of customizing the Endpoint in the MegaWizard Plug-In Manager design flow. It specifies the same options that are chosen in Chapter 14, Testbench and Design Example.

For further information about the parameter settings, refer to Chapter 3, Parameter Settings.

Follow these steps to customize your variant in the MegaWizard Plug-In Manager:

1. On the Tools menu, click **MegaWizard Plug-In Manager**. The MegaWizard Plug-In Manager appears.
2. Select **Create a new custom megafunction variation** and click **Next**.
3. In **Which device family will you be using?** Select the **Arria V** device family.
4. Expand the **Interfaces** directory under **Installed Plug-Ins** by clicking the + icon left of the directory name, expand **PCI Express**, then click **Arria V Hard IP for PCI Express** <*version_number*>
5. Select the output file type for your design. This walkthrough supports VHDL and Verilog HDL. For this example, select **Verilog HDL**.
6. Specify a variation name for output files <*working_dir*>/**example_design/**<*variation name*>. For this walkthrough, specify <*working_dir*>/**example_design/gen1_x4.**
7. Click **Next** to open the parameter editor for the **Arria V Hard IP for PCI Express**.
8. Specify the **System Settings** values listed inTable 2–1.

Table 2–1. System Settings Parameters

Parameter	Value
Number of Lanes	×4
Lane Rate	Gen 1 (2.5 Gbps)
Port type	Native endpoint
Application Layer interface	Avalon-ST 64-bit
RX buffer credit allocation - performance for received requests	Low
Reference clock frequency	100 MHz
Use 62.5 MHz Application Layer clock for ×1	Leave this option off
Number of functions	1

Each function shares the parameter settings on the **Device**, **Error Reporting**, **Link**, **Slot**, and **Power Management** tabs. Each function has separate parameter settings for the **Base Address Registers**, **Base and Limit Registers for Root Ports**, **Device Identification Registers**, and the **PCI Express/PCI Capabilites** parameters. When you click on a **Func**<*n*> tab under the **Port Functions** heading, the tabs automatically reflect the **Func**<*n*> tab selected.

9. Specify the **Device** parameters listed in Table 2–2.

Table 2–2. Device

Parameter	Value
Maximum payload size	256 bytes
Number of tags supported	32
Completion timeout range	ABCD
Implement completion timeout disable	On

10. On the **Error Reporting** tab, leave all options off.
11. Specify the **Link** settings listed in Table 2–7.

Table 2–3. Link Tab

Parameter	Value
Link port number	1
Slot clock configuration	On

12. On the **Slot Capabilities** tab, leave the **Slot register** turned off.
13. Specify the **Power Management** parameters listed in Table 2–4.

Table 2–4. Power Management Parameters

Parameter	Value
Endpoint L0s acceptable exit latency	Maximum of 64 ns
Endpoint L1 acceptable latency	Maximum of 1 µs

14. Specify the **BAR** settings for **Func0** listed in Table 2–5.

Table 2–5. Base Address Registers for Func0

Parameter	Value
BAR0 Type	64-bit prefetchable memory
BAR0 Size	256 MBytes - 28 bits
BAR1 Type	Disabled
BAR1 Size	N/A
BAR2 Type	32-bit non-prefetchable memory
BAR2 Size	1 KByte - 10 bits

15. You can leave **Func0 BAR3** through **Func0 BAR5** and the **Func0 Expansion ROM Disabled**.

16. Under the **Base and Limit Registers** heading, disable both the **Input/Output** and **Prefetchable memory** options. (These options are for Root Ports.)

17. Specify the **Device ID Registers for Func0** listed in Table 2–6.

Table 2–6. Device ID Registers for Func0

Register Name	Value
Vendor ID	0x00000000
Device ID	0x00000001
Revision ID	0x00000001
Class Code	0x00000000
Subsystem Vendor ID	0x00000000
Subsystem Device ID	0x00000000

18. On the **Func 0 Device** tab, under **PCI Express/PCI Capabilities for Func 0** turn **Function Level Reset (FLR) On**.

19. Table 2–7 lists settings for the **Func0 Link** tab.

Table 2–7. Link Capabilities

Parameter	Value
Data link layer active reporting	Off
Surprise down reporting	Off

20. On the **Func0 MSI** tab, for **Number of MSI messages requested**, select **4**.

21. On the **Func0 MSI-X** tab, turn **Implement MSI-X** off.

22. Click **Finish**. The Generation dialog box appears.

23. Turn on **Generate Example Design** to generate the Endpoint, testbench, and supporting files.

24. Click **Exit**.

25. Click **Yes** if you are prompted to add the Quartus II IP File (**.qip**) to the project.

    The **.qip** is a file generated by the parameter editor contains all of the necessary assignments and information required to process the IP core in the Quartus II compiler. Generally, a single **.qip** file is generated for each IP core.

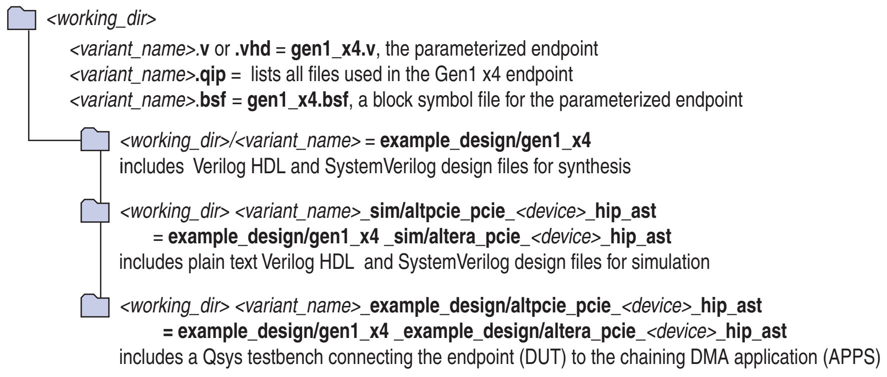
Understanding the Files Generated

Figure 2–2 illustrates the directory structure created for this design after you generate the Arria V Hard IP for PCI Express. Generation creates three directories:

- *<working_dir>*/*<variant_name>* includes the files for synthesis.
- *<working_dir>*/*<variant_name>***_sim/altera_pcie_av_hip_ast** includes the simulation files.
- *<working_dir>*/*<variant_name>***_example_design/altera_pcie_av_hip_ast** contains a Qsys system that connects the Endpoint variant to a chaining DMA design example for verification.

Figure 2–2 illustrates this directory structure.

Figure 2–2. Directory Structure for Arria V Hard IP for PCI Express IP Simulation Model and Design Example



<working_dir>

<variant_name>.**v** or **.vhd** = **gen1_x4.v**, the parameterized endpoint
<variant_name>.**qip** = lists all files used in the Gen1 x4 endpoint
<variant_name>.**bsf** = **gen1_x4.bsf**, a block symbol file for the parameterized endpoint

<working_dir>/<variant_name> = **example_design/gen1_x4**
includes Verilog HDL and SystemVerilog design files for synthesis

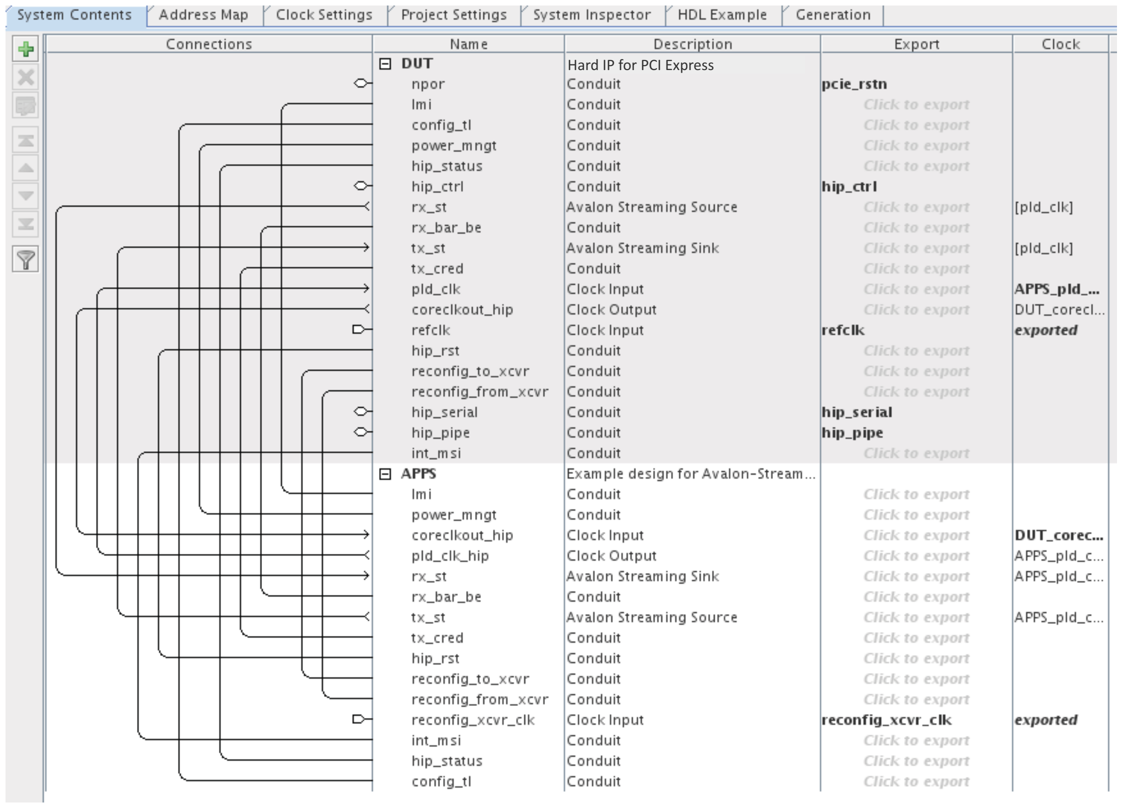
*<working_dir> <variant_name>*_**sim/altpcie_pcie**_*<device>*_**hip_ast**
= **example_design/gen1_x4 _sim/altera_pcie**_*<device>*_**hip_ast**
includes plain text Verilog HDL and SystemVerilog design files for simulation

*<working_dir> <variant_name>*_**example_design/altpcie_pcie**_*<device>*_**hip_ast**
= **example_design/gen1_x4 _example_design/altera_pcie**_*<device>*_**hip_ast**
includes a Qsys testbench connecting the endpoint (DUT) to the chaining DMA application (APPS)

Follow these steps to generate the chaining DMA testbench from the Qsys system design example.

1. On the Quartus II File menu, click **Open**.
2. Navigate to the Qsys system in the **altera_pcie_sv_hip_ast** subdirectory.
3. Click **pcie_de_gen1_x4_ast64.qsys** to bring up the Qsys design. Figure 2–2 illustrates this Qsys system.

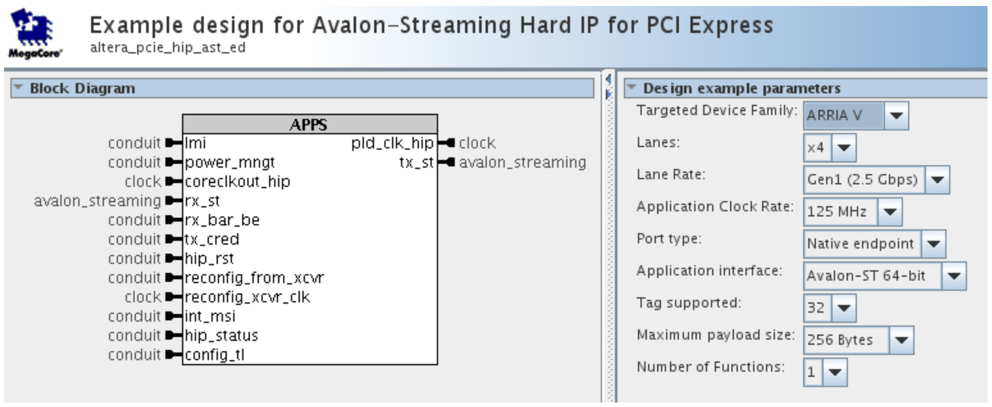
Figure 2–3. Qsys System Connecting the Endpoint Variant and Chaining DMA Testbench



Connections	Name	Description	Export	Clock
	⊟ **DUT**	Hard IP for PCI Express		
	npor	Conduit	**pcie_rstn**	
	lmi	Conduit	Click to export	
	config_tl	Conduit	Click to export	
	power_mngt	Conduit	Click to export	
	hip_status	Conduit	Click to export	
	hip_ctrl	Conduit	**hip_ctrl**	
	rx_st	Avalon Streaming Source	Click to export	[pld_clk]
	rx_bar_be	Conduit	Click to export	
	tx_st	Avalon Streaming Sink	Click to export	[pld_clk]
	tx_cred	Conduit	Click to export	
	pld_clk	Clock Input	Click to export	**APPS_pld_...**
	coreclkout_hip	Clock Output	Click to export	DUT_corecl...
	refclk	Clock Input	**refclk**	***exported***
	hip_rst	Conduit	Click to export	
	reconfig_to_xcvr	Conduit	Click to export	
	reconfig_from_xcvr	Conduit	Click to export	
	hip_serial	Conduit	**hip_serial**	
	hip_pipe	Conduit	**hip_pipe**	
	int_msi	Conduit	Click to export	
	⊟ **APPS**	Example design for Avalon-Stream...		
	lmi	Conduit	Click to export	
	power_mngt	Conduit	Click to export	
	coreclkout_hip	Clock Input	Click to export	**DUT_corec...**
	pld_clk_hip	Clock Output	Click to export	APPS_pld_c...
	rx_st	Avalon Streaming Sink	Click to export	APPS_pld_c...
	rx_bar_be	Conduit	Click to export	
	tx_st	Avalon Streaming Source	Click to export	APPS_pld_c...
	tx_cred	Conduit	Click to export	
	hip_rst	Conduit	Click to export	
	reconfig_to_xcvr	Conduit	Click to export	
	reconfig_from_xcvr	Conduit	Click to export	
	reconfig_xcvr_clk	Clock Input	**reconfig_xcvr_clk**	***exported***
	int_msi	Conduit	Click to export	
	hip_status	Conduit	Click to export	
	config_tl	Conduit	Click to export	

4. To display the parameters of the **APPS** component shown in Figure 2–2, click on it and then select **Edit** from the right-mouse menu. Figure 2–3 illustrates this component. Note that the values for the following parameters match those set in the DUT component:
   - **Targeted Device Family**
   - **Lanes**
   - **Lane Rate**
   - **Application Clock Rate**
   - **Port type**
   - **Application interface**
   - **Tags supported**
   - **Maximum payload size**
   - **Number of Functions**

Figure 2–4. Qsys Component Representing the Chaining DMA Design Example



Targeted Device Family: ARRIA V
Lanes: x4
Lane Rate: Gen1 (2.5 Gbps)
Application Clock Rate: 125 MHz
Port type: Native endpoint
Application interface: Avalon-ST 64-bit
Tag supported: 32
Maximum payload size: 256 Bytes
Number of Functions: 1

☞ You can use this Qsys APPS component to test any Endpoint variant with the same values for these parameters.

5. To close the **APPS** component, click the **X** in the upper right-hand corner of the parameter editor.

Go to "Generating the Simulation Model Using Qsys" on page 2–16 for instructions on system simulation.

Qsys Design Flow

This section guides you through the steps necessary to customize the Arria V Hard IP for PCI Express and run the example testbench in Qsys. It includes the following steps:

- Customizing the Endpoint in Qsys
- Understanding the Files Generated
- Generating the Simulation Model Using Qsys
- Compiling the Design in the Qsys Design Flow
- Modifying the Example Design

Customizing the Endpoint in Qsys

This section begins with the steps necessary to customize the Arria V Hard IP for PCI Express. This section also guides you through steps to connect the chaining DMA component testbench.

For further details about the parameter settings, refer to Chapter 3, Parameter Settings.

Follow these steps to instantiate the Arria V Hard IP for PCI Express and chaining DMA example design using the Qsys design flow:

1. Create a directory for your project. This example uses <*working_dir*>**/pcie_qsys.**
2. To start Qsys from the Quartus II software, on the File menu click **New**.
3. In the **New** dialog box, click **Qsys System File**, then click **OK**. Qsys appears.
4. On the Project Settings tab specify the settings listed in Table 2–8

Table 2–8. Project Settings Parameters

Parameter	Value
Device family	Arria V
Clock crossing adapter type	Handshake
Limit interconnect pipeline stages to (1)	0
Generation ID	0

Note to Table 2–8:

(1) This setting applies to designs that include Avalon-MM interfaces.

5. On the **Component Library** tab, type the following text string in the search box:

```
PCI Ex ↵
```

   Qsys filters the component library and shows all components matching the text string you entered.

6. Click on **Arria V Hard IP for PCI Express** and then click the +**Add** button. The parameter editor appears.

The following sections provide step-by-step instructions to create the example design in Qsys. If you prefer, you can copy the completed system from the Quartus II software installation, and then go to "Generating the Simulation Model Using Qsys" on page 2–16. The completed Qsys systems are located in the following directory: <*install_dir*>**/ip/altera/altera_pcie/altera_pcie_hip_ast_ed/example_design/av**.

Specifying the Parameters for the Arria V Hard IP for PCI Express

This section guides you through the process of specifying parameters for the Arria V Hard IP for PCI Express to create a Gen1 ×4 Endpoint.

1. Specify the **System Settings** parameters listed in Table 2–9.

Table 2–9. System Settings Parameters

Parameter	Value
Number of Lanes	×4
Lane Rate	Gen 1 (2.5 Gbps)
Port type	Native endpoint
Application interface	Avalon-ST 64-bit
RX buffer credit allocation - performance for received requests	Low
Reference clock frequency	100 MHz
Use 62.5 MHz Application Layer clock for ×1	Leave this option **Off**
Use deprecated RX Avalon-ST data byte enable port (rx_st_be)	Leave this option **On**.
Number of functions	1

Each function shares the parameter settings on the **Device**, **Error Reporting**, **Link**, **Slot**, and **Power Management** tabs. Each function has separate parameter settings for the **Base Address Registers**, **Base and Limit Registers for Root Ports**, **Device Identification Registers**, and the **PCI Express/PCI Capabilites** parameters. When you click on a **Func<*n*>** tab under the **Port Functions** heading, the tabs automatically reflect the **Func<*n*>** tab selected.

2. Specify the **Device** parameters listed in Table 2–10.

Table 2–10. Device

Parameter	Value
Maximum payload size	256 bytes
Number of tags supported	32
Completion timeout range	ABCD
Implement completion timeout disable	On

3. On the **Error Reporting** tab, leave all options off.

4. Specify the **Link** settings listed in Table 2–11.

Table 2–11. Link Tab

Parameter	Value
Link port number	1
Slot clock configuration	Enabled

5. Specify the **Slot** settings listed in Table 2–12.

Table 2–12. Link Tab

Parameter	Value
Use slot register	Leave this option off.
Data link layer active reporting	0
Surprise down reporting	0
Slot clock configuration	0

6. Specify the **Power Management** settings listed in.Table 2–13.

Table 2–13. Power Management Parameters

Parameter	Value
Endpoint L0s acceptable exit latency	Maximum of 64 ns
Endpoint L1 acceptable latency	Maximum of 1 µs

7. Specify the **BAR** settings for **Func0** listed in Table 2–14.

Table 2–14. Base Address Registers for Func0

Parameter	Value
BAR0 Type	64-bit prefetchable memory
BAR0 Size	256 MBytes - 28 bits
BAR1 Type	Disabled
BAR1 Size	N/A
BAR2 Type	32-bit non-prefetchable memory
BAR2 Size	1 KByte - 10 bits

8. You can leave **Func0 BAR3** through **Func0 BAR5** and the **Func0 Expansion ROM Disabled**.
9. Under the **Base and Limit Registers** heading, disable both the **Input/Output** and **Prefetchable memory** options. (These options are for Root Ports.)
10. Specify the **Device ID Registers for Func0** listed in Table 2–15.

Table 2–15. Device Identification Registers for Func0 (Part 1 of 2)

Register Name	Value
Vendor ID	`0x00000000`
Device ID	`0x00000001`
Revision ID	`0x00000001`

Table 2–15. Device Identification Registers for Func0 (Part 2 of 2)

Class Code	`0x00000000`
Subsystem Vendor ID	`0x00000000`
Subsystem Device ID	`0x00000000`

11. On the **Func 0 Device** tab, under **PCI Express/PCI Capabilities for Func 0** turn **Function Level Reset (FLR) On**.
12. Specify the **Link** settings listed in Table 2–16.

Table 2–16. Link Capabilities

Parameter	Value
Data link layer active reporting	Off
Surprise down reporting	Off

13. On the **Func0 MSI** tab, for **Number of MSI messages requested**, select **4**.
14. On the **Func0 MSI-X** tab, turn **Implement MSI-X** turned off.
15. Click the **Finish** button.
16. To rename the **Arria V hard IP for PCI Express**, in the **Name** column of the **System Contents** tab, right-click on the component name, select **Rename**, and type `DUT` ↵

Specifying the Parameters for the Example Design

Follow these steps to add the Example design for Avalon-Streaming Hard IP for PCI Express component to your Qsys system.

1. On the **Component Library** tab, click **Example design for Avalon-Streaming Hard IP for PCI Express** and then click **Add**. The parameter editor appears.
2. Change the parameters to match those of the Gen1 ×4 Endpoint variant by selecting the parameter values shown in Table 2–17.

Table 2–17. Parameters for the Example Design

Parameter	Value
Targeted device family	Arria V
Lanes	×4
Lane rate	Gen1 (2.5 Gbps)
Application Clock Rate	125 MHz
Port type	Native Endpoint
Application interface	Avalon-ST 64-bit
Tags supported	32
Maximum payload size	256
Number of functions	1

3. Click **Finish**.

4. To rename the Example design for Avalon-Streaming Arria V hard IP for PCI Express component, right-click on the component name, select **Rename**, and type `APPS`↵

Completing the Qsys System

The **APPS** component interfaces connect to the Endpoint variant interfaces with matching names. Most of these interfaces are of the Avalon Conduit type, which is a point-to-point interface type that accommodates individual signals or groups of signals that do not fit into any of the other Avalon types. You can connect conduit interfaces to each other inside a Qsys system or export them to make connections to other modules in the design or to FPGA pins.

For more information about Avalon interfaces, refer to the *Avalon Interface Specifications*.

Follow these steps to export Avalon Conduit interfaces that connect outside the Qsys system.

1. To export the `npor` interface which is a power-on reset pin for the FPGA, click in the **Export** column and type `dut_npor` which is the name of the exported interface. Note that the **Connections** column now shows that the `npor` interface is no longer available for internal connections in Qsys.
2. Export the following interfaces, using the same name in the **Export** Column that is shown in the **Name** column of Qsys.
   - `dut_hip_ctrl`
   - `dut_refclk`
   - `dut_hip_serial`
   - `dut_hip_pipe`
   - `reconfig_xcvr_clk` (This interface is part of the APPS component)

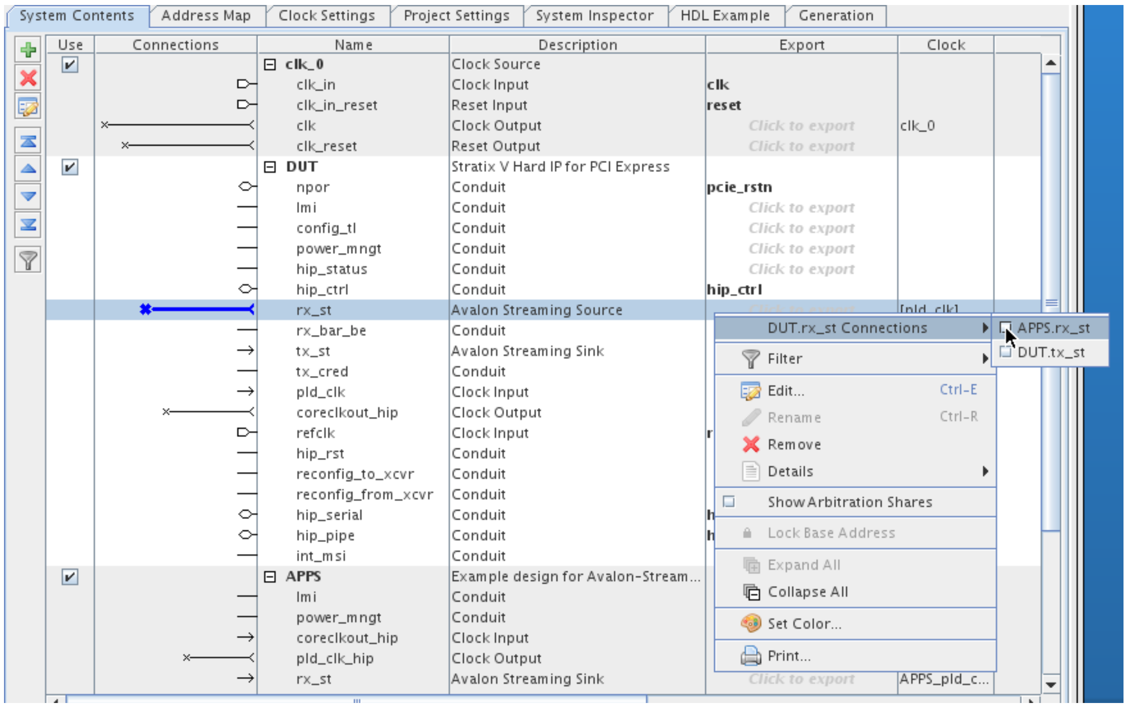
   You can select **Undo Export Interface** on the Edit menu if you accidentally export an interface.

In this example design the Avalon-ST source interface of the **DUT** connects to the Avalon-ST sink interface of the **APPS** component, and the Avalon-ST sink interface of the **DUT**, connects to the Avalon-ST source interface of the **APPS** component. Follow these steps to connect the Avalon-ST source and sink interfaces of these two components:

1. To connect the Avalon-ST `rx_st` source interface of the DUT component to the Avalon-ST `rx_st` sink interface of the APPS, in the **Name** column, right-click on the `rx_st` interface and select and select **Apps.rx_st** from the **DUT.rx_st Connections** list.

   Figure 2–4 illustrates this procedure.

Figure 2–5. Connecting Signals Using the Right-Mouse Connections Menu



Use	Connections	Name	Description	Export	Clock
✓		⊟ **clk_0**	Clock Source		
		clk_in	Clock Input	**clk**	
		clk_in_reset	Reset Input	**reset**	
		clk	Clock Output	Click to export	clk_0
		clk_reset	Reset Output	Click to export	
✓		⊟ **DUT**	Stratix V Hard IP for PCI Express		
		npor	Conduit	**pcie_rstn**	
		lmi	Conduit	Click to export	
		config_tl	Conduit	Click to export	
		power_mngt	Conduit	Click to export	
		hip_status	Conduit	Click to export	
		hip_ctrl	Conduit	**hip_ctrl**	
		rx_st	Avalon Streaming Source		[pld_clk]
		rx_bar_be	Conduit		
		tx_st	Avalon Streaming Sink		
		tx_cred	Conduit		
		pld_clk	Clock Input		
		coreclkout_hip	Clock Output		
		refclk	Clock Input	r	
		hip_rst	Conduit		
		reconfig_to_xcvr	Conduit		
		reconfig_from_xcvr	Conduit		
		hip_serial	Conduit	h	
		hip_pipe	Conduit	h	
		int_msi	Conduit		
✓		⊟ **APPS**	Example design for Avalon-Stream...		
		lmi	Conduit		
		power_mngt	Conduit		
		coreclkout_hip	Clock Input		
		pld_clk_hip	Clock Output		
		rx_st	Avalon Streaming Sink	Click to export	APPS_pld_c...

2. To connect the Avalon-ST `tx_st` source interface of the APPS component to the `tx_st` sink interface of the DUT component, repeat the technique explained in Step 1.

For conduit interface types, the **APPS** component interfaces connect to the **DUT** interfaces with *matching* names.

☞ The Avalon Conduit interface type is a point-to-point interface type that accommodates individual signals or groups of signals that do not fit into any of the other Avalon types. You can connect conduit interfaces to each other inside a Qsys system or export them to make connections to other modules in the design or to FPGA pins.

For more information about Avalon interfaces, refer to the *Avalon Interface Specifications*.

3. Connect the following Avalon Conduit interfaces using the technique described in Step 1.
   - `lmi`
   - `config_tl`
   - `power_mgmt`
   - `hip_status`
   - `rx_bar_be`
   - `tx_cred`
   - `hip_rst`
   - `reconfig_to_xcvr`
   - `reconfig_from_xcvr`
   - `int_msi`
4. Follow these steps to connect the clocks:
   a. In the **Clock** column right-click on the DUT `pld_clk` interface and select **APPS.pld_clk_hip** from the **DUT.pld_clk Connections** list.

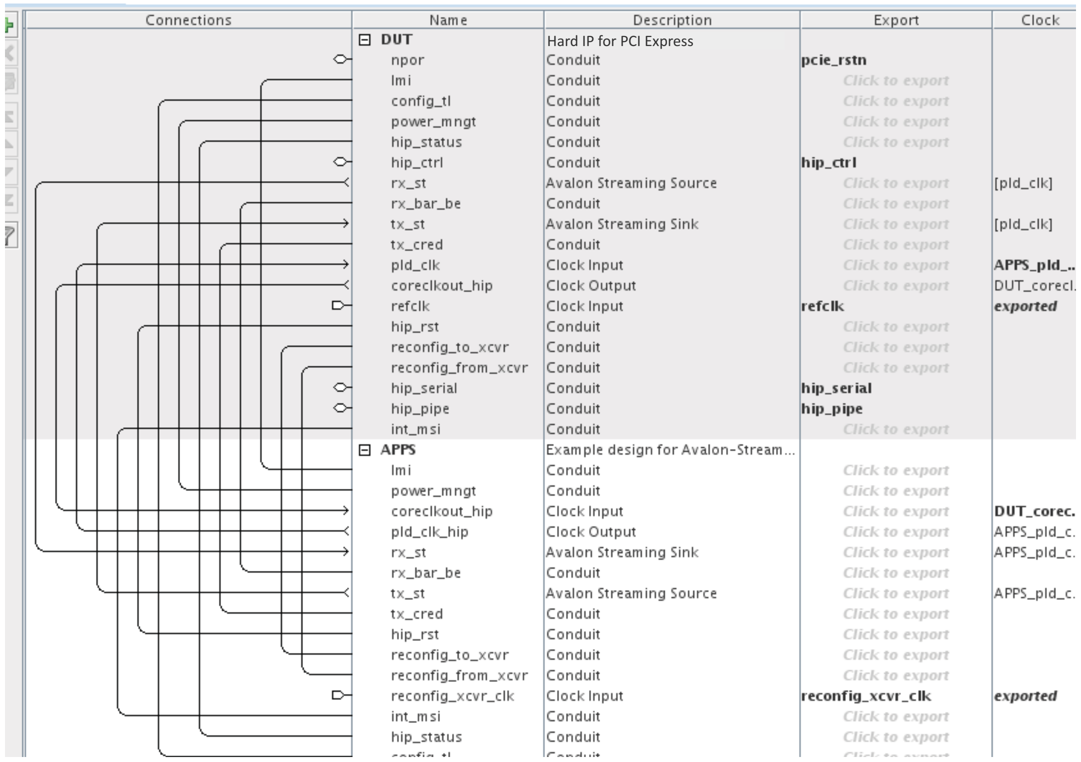
   b. To connect the APPS `pld_clk_hip` interface to the DUT `pld_clk` interface, right-click on **APPS.pld_clk_hip** and select **DUT.pld_clk** from the **APPS.pld_clk_hip Connections** list.

   c. To connect the DUT `coreclkout_hip` interface to the APPS `coreclkout_hip` interface, right-click on **DUT.coreclkout_hip** and select **DUT.coreclkout_hip** from the **DUT.coreclkout_hip Connections** list. CHECK

   d. To connect the DUT `coreclkout_hip` interface to the APPS `coreclkout_hip` interface, right-click on **DUT.coreclkout_hip** and select **APPS.coreclkout_hip** from the **DUT.coreclkout_hip Connections** list.
5. To remove the default clock, on the **System Contents** tab, click **clk_0** and then click the X button.
6. To save your Qsys system, on the File menu select **Save**. Type `pcie_qsys` in the Save dialog box.

Figure 2–5 illustrates the complete Qsys system.

Figure 2–6. Complete Gen1 ×4 Endpoint (DUT) Connected to Example Design (APPS)



Connections	Name	Description	Export	Clock
	⊟ DUT	Hard IP for PCI Express		
	npor	Conduit	pcie_rstn	
	lmi	Conduit	Click to export	
	config_tl	Conduit	Click to export	
	power_mngt	Conduit	Click to export	
	hip_status	Conduit	Click to export	
	hip_ctrl	Conduit	hip_ctrl	
	rx_st	Avalon Streaming Source	Click to export	[pld_clk]
	rx_bar_be	Conduit	Click to export	
	tx_st	Avalon Streaming Sink	Click to export	[pld_clk]
	tx_cred	Conduit	Click to export	
	pld_clk	Clock Input	Click to export	APPS_pld_..
	coreclkout_hip	Clock Output	Click to export	DUT_corecl.
	refclk	Clock Input	refclk	*exported*
	hip_rst	Conduit	Click to export	
	reconfig_to_xcvr	Conduit	Click to export	
	reconfig_from_xcvr	Conduit	Click to export	
	hip_serial	Conduit	hip_serial	
	hip_pipe	Conduit	hip_pipe	
	int_msi	Conduit	Click to export	
	⊟ APPS	Example design for Avalon-Stream...		
	lmi	Conduit	Click to export	
	power_mngt	Conduit	Click to export	
	coreclkout_hip	Clock Input	Click to export	DUT_corec.
	pld_clk_hip	Clock Output	Click to export	APPS_pld_c.
	rx_st	Avalon Streaming Sink	Click to export	APPS_pld_c.
	rx_bar_be	Conduit	Click to export	
	tx_st	Avalon Streaming Source	Click to export	APPS_pld_c.
	tx_cred	Conduit	Click to export	
	hip_rst	Conduit	Click to export	
	reconfig_to_xcvr	Conduit	Click to export	
	reconfig_from_xcvr	Conduit	Click to export	
	reconfig_xcvr_clk	Clock Input	reconfig_xcvr_clk	*exported*
	int_msi	Conduit	Click to export	
	hip_status	Conduit	Click to export	

Generating the Simulation Model Using Qsys

Follow these steps to generate a simulation model that you can include in your own PCI Express testbench.

1. On the Qsys **Generation** tab, specify the parameters listed in Table 2–18.

Table 2–18. Parameters to Specify on the Generation Tab in Qsys (Part 1 of 2)

Parameter	Value
Simulation	
Create simulation model	**Verilog**
Create testbench Qsys system (1)	**Standard, BFMs for standard Avalon interfaces or None**
Create testbench simulation model (1)	**Verilog or None**
Synthesis	
Create HDL design files for synthesis	Turn on this option
Create block symbol file (.bsf)	Turn on this option
Output Directory	
Path	**pcie_qsys/pcie_de_gen1_x4_ast64**
Simulation	**pcie_qsys/pcie_de_gen1_x4_ast64/simulation**
Testbench	**pcie_qsys/pcie_de_gen1_x4_ast64/testbench**

Table 2–18. Parameters to Specify on the Generation Tab in Qsys (Part 2 of 2)

Parameter	Value
Synthesis	pcie_qsys/pcie_de_gen1_x4_ast64/t/synthesis

Note to Table 2–18:

(1) You can generate a simulation testbench; however the Root Port BFM is not available in the current release. Simulation models are available for Verilog HDL.

2. Click the **Generate** button at the bottom of the **Generation** tab to create the chaining DMA simulation model, which you can include in your own custom testbench.

☞ You can also generate the testbench which shows you the connections necessary between the Endpoint and a Root Port BFM. However, the Root Port BFM is not available for simulation in the current release.

Quartus II Compilation

This section provides step-by-step instructions for Quartus II compilation. To compile your Endpoint and design example, complete the instructions in one of the following two sections:

- Compiling the Design in the MegaWizard Plug-In Manager Design Flow
- Compiling the Design in the Qsys Design Flow

Compiling the Design in the MegaWizard Plug-In Manager Design Flow

To compile your design, on the Processing menu, select **Start Compilation**.

Compiling the Design in the Qsys Design Flow

To compile the Qsys design example in the Quartus II software, you must create a Quartus II project and add your Qsys files to that project.

Complete the following steps to create your Quartus II project:

1. Choose **Programs > Altera > Quartus II** <*version*> (Windows Start menu) to run the Quartus II software.
2. Change to the directory that includes your Qsys project, <*working_dir*>**\pcie_qsys**.
3. On the Quartus II File menu, click **New,** then **New Quartus II Project**, then **OK**.
4. Click **Next** in the **New Project Wizard: Introduction** (The introduction does not display if you previously turned it off.)
5. On the **Directory, Name, Top-Level Entity** page, enter the following information:
    a. The working directory for your project. This design example uses <*working_dir*>**/pcie_qsys**
    b. The name of the project. Type the same name as your Qsys design `pcie_de_gen1_x4_ast64` ↵

    ☞ If the top-level design entity and Qsys system names are identical, the Quartus II software treats the Qsys system as the top-level design entity.

6. Click **Next** to display the **Add Files** page.
7. Perform the following steps to add the Quartus II IP File (**.qip**) to the project:
   a. Click the browse button next the **File name** box and browse to **pcie_de_gen1_x4_ast64/synthesis/** directory.
   b. In the **Files of type** list, select **IP Variation Files (*.qip)**.
   c. Click **pcie_de_gen1_x4_ast64.qip** and then click **Open**.
   d. On the **Add Files** page, click **Add**, then click **OK**.

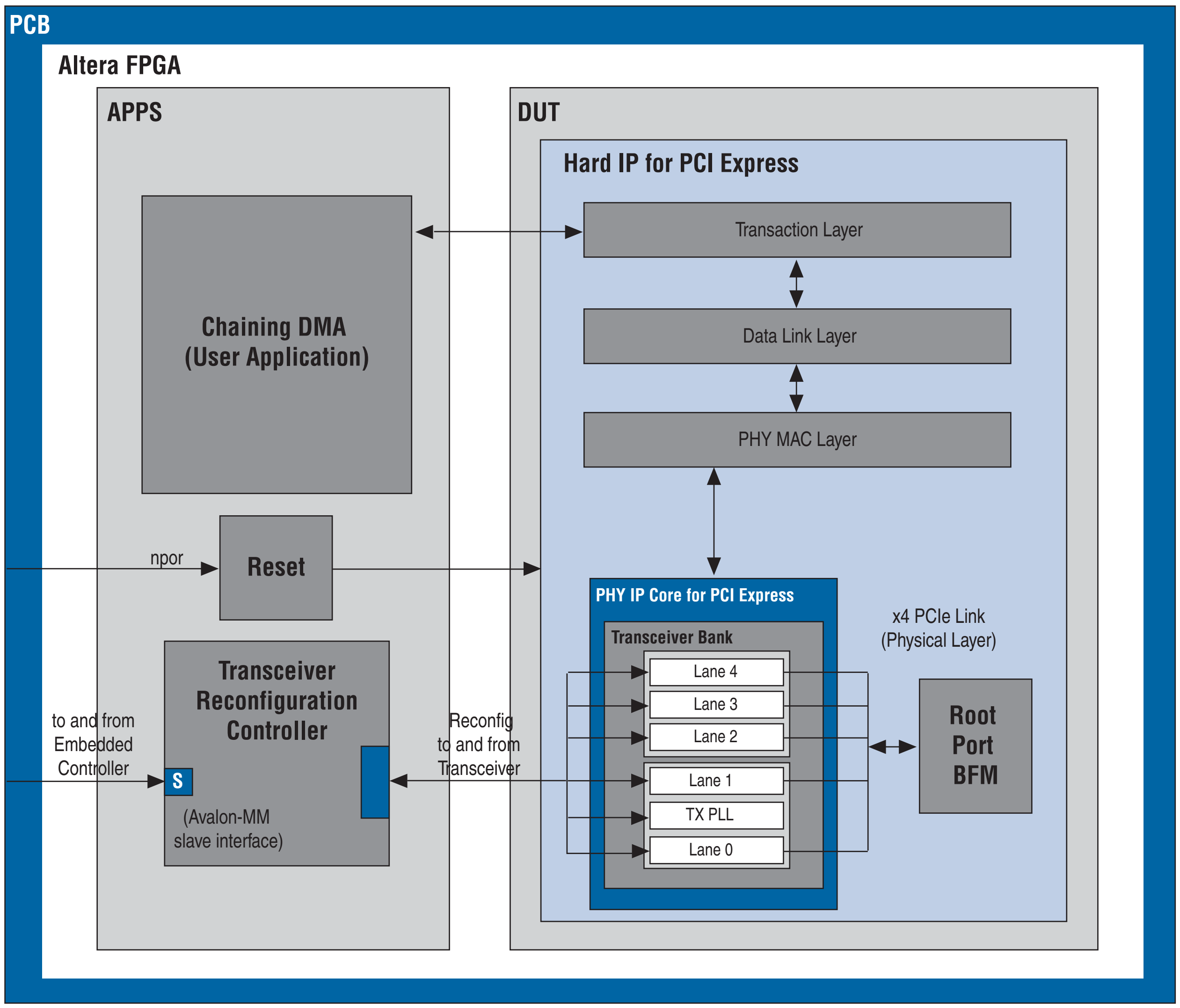
   Click **Yes**, if prompted, to create a new directory.

8. Click **Next** to display the **Device** page.
9. On the **Family & Device Settings** page, choose the following target device family and options:
   a. In the **Family** list, select **Arria V**.
   b. In the **Devices** list, select **Arria V GX PCIe**.
   c. In the **Available devices** list, select **5AGXFB3H6F35C6ES**.
10. Click **Next** to close this page and display the **EDA Tool Settings** page.
11. Click **Next** to display the **Summary** page.
12. Check the **Summary** page to ensure that you have entered all the information correctly.
13. Click **Finish** to create the Quartus II project.
14. To compile your design using the Quartus II software, on the Processing menu, click **Start Compilation**. The Quartus II software then performs all the steps necessary to compile your design.

Modifying the Example Design

To use this example design as the basis of your own design, replace the Chaining DMA Example shown in Figure 2–6 with your own Application Layer design. Then, create a Root Port BFM driver to generate the transactions needed to test your Application Layer.

Figure 2–7. Testbench for PCI Express


PCB
Altera FPGA
APPS
DUT
Hard IP for PCI Express
Transaction Layer
Data Link Layer
PHY MAC Layer
Chaining DMA (User Application)
npor
Reset
Transceiver Reconfiguration Controller
to and from Embedded Controller
S
(Avalon-MM slave interface)
Reconfig to and from Transceiver
PHY IP Core for PCI Express
Transceiver Bank
Lane 4
Lane 3
Lane 2
Lane 1
TX PLL
Lane 0
x4 PCIe Link (Physical Layer)
Root Port BFM

3. Parameter Settings

This chapter describes the parameters which you can set using the MegaWizard Plug-In Manager or Qsys design flow to instantiate a Arria V Hard IP for PCI Express IP core. The appearance of the GUI is identical for the two design flows.

☞ In the following tables, hexadecimal addresses in green are links to additional information in the "Register Descriptions" chapter.

System Settings

The first group of settings defines the overall system. Table 3–1 describes these settings.

Table 3–1. System Settings for PCI Express (Part 1 of 3)

Parameter	Value	Description
Number of Lanes	**×1, ×4, ×8**	Specifies the maximum number of lanes supported.
Lane Rate	**Gen1 (2.5 Gbps)** **Gen2 (5.0 Gbps)**	Specifies the maximum data rate at which the link can operate.
Port type	**Native Endpoint** **Root Port** **Legacy Endpoint**	Specifies the function of the port. Altera recommends **Native Endpoint** for all new Endpoint designs. Select **Legacy Endpoint** only when you require I/O transaction support for compatibility. The Endpoint stores parameters in the Type 0 Configuration Space which is outlined in Table 6–2 on page 6–2. The Root Port stores parameters in the Type 1 Configuration Space which is outlined in Table 6–3 on page 6–2.
Application Interface	**64-bit Avalon-ST** **128-bit Avalon-ST**	Specifies the interface between the PCI Express Transaction Layer and the Application Layer. Refer to Table 7–1 on page 7–3 for a comprehensive list of available link width, interface width, and frequency combinations.

Table 3–1. System Settings for PCI Express (Part 2 of 3)

Parameter	Value	Description
RX Buffer credit allocation - performance for received requests	**Minimum** **Low** **Medium** **High** **Maximum**	This setting determines the allocation of posted header credits, posted data credits, non-posted header credits, completion header credits, and completion data credits in the 6 KByte RX buffer. The 5 settings allow you to adjust the credit allocation to optimize your system. The credit allocation for the selected setting displays in the message pane. Refer to Chapter 11, Flow Control, for more information about optimizing performance. The Flow Control chapter explains how the **RX credit allocation** and the **Maximum payload size** that you choose affect the allocation of flow control credits. You can set the **Maximum payload size** parameter in Table 3–2 on page 3–4. ■ **Minimum**–This setting configures the minimum PCIe specification allowed non-posted and posted request credits, leaving most of the RX Buffer space for received completion header and data. Select this option for variations where application logic generates many read requests and only infrequently receives single requests from the PCIe link. ■ **Low**– This setting configures a slightly larger amount of RX Buffer space for non-posted and posted request credits, but still dedicates most of the space for received completion header and data. Select this option for variations where application logic generates many read requests and infrequently receives small bursts of requests from the PCIe link. This option is recommended for typical endpoint applications where most of the PCIe traffic is generated by a DMA engine that is located in the endpoint application layer logic. ■ **Medium**–This setting allocates approximately half the RX Buffer space to received requests and the other half of the RX Buffer space to received completions. Select this option for variations where the received requests and received completions are roughly equal. ■ **High**–This setting configures most of the RX Buffer space for received requests and allocates a slightly larger than minimum amount of space for received completions. Select this option where most of the PCIe requests are generated by the other end of the PCIe link and the local application layer logic only infrequently generates a small burst of read requests. This option is recommended for typical root port applications where most of the PCIe traffic is generated by DMA engines located in the endpoints. ■ **Maximum**–This setting configures the minimum PCIe specification allowed amount of completion space, leaving most of the RX Buffer space for received requests. Select this option when most of the PCIe requests are generated by the other end of the PCIe link and the local application layer logic never or only infrequently generates single read requests. This option is recommended for control and status endpoint applications that don't generate any PCIe requests of their own and only are the target of write and read requests from the root complex.

Table 3–1. System Settings for PCI Express (Part 3 of 3)

Parameter	Value	Description
Reference clock frequency	**100 MHz** **125 MHz**	The *PCI Express Base Specification 2.1* requires a 100 MHz ±300 ppm reference clock. The 125 MHz reference clock is provided as a convenience for systems that include a 125 MHz clock source.
Use 62.5 MHz Application Layer clock	**On/Off**	This is a special power saving mode available only for Gen1 ×1 variants.
Use deprecated RX Avalon-ST data byte enable port (rx_st_be)	**On/Off**	When enabled the variant includes the deprecated `rx_st_be` signals.
Number of functions	**1–8**	Specifies the number of functions that share the same link.

Port Functions

This section describes the parameter settings for port functions. It includes the following sections:

- Parameters Shared Across All Port Functions
- Parameters Defined Separately for All Port Functions

Parameters Shared Across All Port Functions

This section defines the PCI Express and PCI capabilities parameters that are shared for all port functions. It includes the following capabilities:

- Device
- Error Reporting
- Link
- Slot
- Power Management

Some of these parameters are stored in the Common Configuration Space Header. Text in green are links to these parameters stored in the Common Configuration Space Header.

Device

Table 3–2 describes the shared device parameters.

Table 3–2. Capabilities Registers for Function <*n*> (Part 1 of 2)

Parameter	Possibl e Values	Default Value	Description
Device Capabilities			
Maximum payload size 0x084	**128 bytes 256 bytes, 512 bytes,**	128 bytes	Specifies the maximum payload size supported. This parameter sets the read-only value of the max payload size supported field of the Device Capabilities register (0x084[2:0]) and optimizes the IP core for this size payload.
Number of tags supported supported per function	**32 64**	32	Indicates the number of tags supported for non-posted requests transmitted by the Application Layer. This parameter sets the values in the Device Control register (0x088) of the PCI Express Capability Structure described in Table 6–8 on page 6–4. The Transaction Layer tracks all outstanding completions for non-posted requests made by the Application Layer. This parameter configures the Transaction Layer for the maximum number to track. The Application Layer must set the tag values in all non-posted PCI Express headers to be less than this value. Values greater than 32 also set the extended tag field supported bit in the `Configuration Space Device Capabilities` register. The Application Layer can only use tag numbers greater than 31 if configuration software sets the `Extended Tag Field Enable` bit of the `Device Control` register. This bit is available to the Application Layer as `cfg_devcsr[8]`.
Completion timeout range	**ABCD BCD ABC AB B A None**	ABCD	Indicates device function support for the optional completion timeout programmability mechanism. This mechanism allows system software to modify the completion timeout value. This field is applicable only to Root Ports and Endpoints that issue requests on their own behalf. Completion timeouts are specified and enabled in the Device Control 2 register (0x0A8) of the PCI Express Capability Structure Version 2.0 described in Table 6–8 on page 6–4. For all other functions this field is reserved and must be hardwired to 0x0000b. Four time value ranges are defined: ■ Range A: 50 µs to 10 ms ■ Range B: 10 ms to 250 ms ■ Range C: 250 ms to 4 s ■ Range D: 4 s to 64 s Bits are set to show timeout value ranges supported. 0x0000b completion timeout programming is not supported and the function must implement a timeout value in the range 50 s to 50 ms.

Table 3–2. Capabilities Registers for Function <*n*> (Part 2 of 2)

Parameter	Possibl e Values	Default Value	Description
Completion timeout range (continued)			The following encodings are used to specify the range: ■ 0001 Range A ■ 0010 Range B ■ 0011 Ranges A and B ■ 0110 Ranges B and C ■ 0111 Ranges A, B, and C ■ 1110 Ranges B, C and D ■ 1111 Ranges A, B, C, and D All other values are reserved. Altera recommends that the completion timeout mechanism expire in no less than 10 ms.
Implement completion timeout disable 0x0A8	**On/Off**	On	For PCI Express version 2.0 and higher Endpoints, this option must be **On**. The timeout range is selectable. When **On**, the core supports the completion timeout disable mechanism via the PCI Express `Device Control Register 2`. The Application Layer logic must implement the actual completion timeout mechanism for the required ranges.

Error Reporting

Table 3–3 describes the Advanced Error Reporting (AER) and ECRC parameters.

Table 3–3. Error Reporting 0x800–0x834

Parameter	Value	Default Value	Description
Advanced error reporting (AER)	**On/Off**	Off	When **On**, enables the AER capability.
ECRC checking	**On/Off**	Off	When **On**, enables ECRC checking. Sets the read-only value of the ECRC check capable bit in the `Advanced Error Capabilities and Control Register`. This parameter requires you to enable the AER capability.
ECRC generation	**On/Off**	Off	When **On**, enables ECRC generation capability. Sets the read-only value of the ECRC generation capable bit in the `Advanced Error Capabilities and Control Register`. This parameter requires you to enable the AER capability.
ECRC forwarding	**On/Off**	Off	When **On**, enables ECRC forwarding to the Application Layer. On the Avalon-ST RX path, the incoming TLP contains the ECRC dword (1) and the `TD` bit is set if an ECRC exists. On the transmit the TLP from the Application Layer must contain the ECRC dword and have the `TD` bit set.

Note to Table 3–3:

(1) Throughout *The Arria V Hard IP for PCI Express User Guide*, the terms word, dword and qword have the same meaning that they have in the *PCI Express Base Specification Revision 2.1 or 3.0*. A word is 16 bits, a dword is 32 bits, and a qword is 64 bits.

Link

Table 3–4 describes the Link Capabilities parameters.

Table 3–4. Link Capabilities 0x090

Parameter	Value	Description
Link port number	**0x01**	Sets the read-only value of the port number field in the `Link Capabilities` register.
Slot clock configuration	**On/Off**	When **On**, indicates that the Endpoint or Root Port uses the same physical reference clock that the system provides on the connector. When **Off**, the IP core uses an independent clock regardless of the presence of a reference clock on the connector.

Slot

Table 3–12 describes the Slot Capabilities parameters.

Table 3–5. Slot Capabilities 0x094

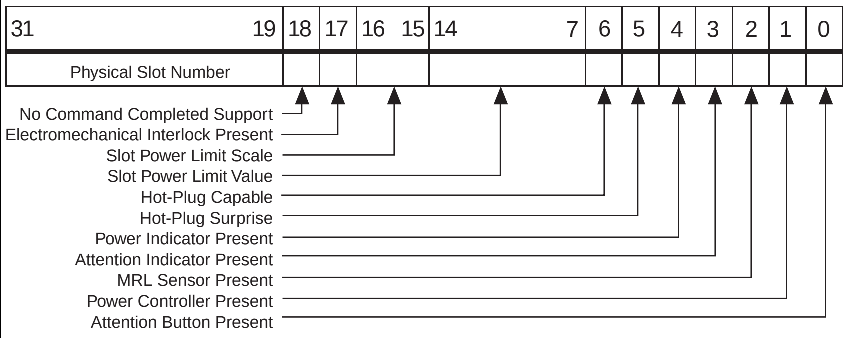
Parameter	Value	Description
Use Slot register	**On/Off**	The slot capability is required for Root Ports if a slot is implemented on the port. Slot status is recorded in the `PCI Express Capabilities Register`. This parameter is only valid for Root Port variants.
Slot Capability register		Defines the characteristics of the slot. You turn this option on by selecting. The various bits are defined as follows:  31 19 18 17 16 15 14 7 6 5 4 3 2 1 0 Physical Slot Number No Command Completed Support Electromechanical Interlock Present Slot Power Limit Scale Slot Power Limit Value Hot-Plug Capable Hot-Plug Surprise Power Indicator Present Attention Indicator Present MRL Sensor Present Power Controller Present Attention Button Present
Slot power scale	`0–3`	Specifies the scale used for the **Slot power limit**. The following coefficients are defined: ■ 0 = 1.0x ■ 1 = 0.1x ■ 2 = 0.01x ■ 3 = 0.001x The default value prior to hardware and firmware initialization is b'00. Writes to this register also cause the port to send the `Set_Slot_Power_Limit` Message. Refer to Section 6.9 of the *PCI Express Base Specification Revision 2.1* for more information.

Table 3–5. Slot Capabilities 0x094

Parameter	Value	Description
Slot power limit	`0–255`	In combination with the **Slot power scale value**, specifies the upper limit in watts on power supplied by the slot. Refer to Section 7.8.9 of the *PCI Express Base Specification Revision 2.1* for more information.
Slot number	`0-8191`	Specifies the slot number.

Power Management

Table 3–6 describes the Power Management parameters.

Table 3–6. Power Management Parameters

Parameter	Value	Description
Endpoint L0s acceptable latency	**< 64 ns – > No limit**	This design parameter specifies the maximum acceptable latency that the device can tolerate to exit the L0s state for any links between the device and the root complex. It sets the read-only value of the Endpoint L0s acceptable latency field of the `Device Capabilities` register **(**0x084**)**. The Arria V Hard IP for PCI Express does not support the L0s or L1 states. However, in a switched system there may be links connected to switches that have L0s and L1 enabled. This parameter is set to allow system configuration software to read the acceptable latencies for all devices in the system and the exit latencies for each link to determine which links can enable Active State Power Management (ASPM). This setting is disabled for Root Ports. The default value of this parameter is 64 ns. This is the safest setting for most designs.
Endpoint L1 acceptable latency	**< 1** µs to **> No limit**	This value indicates the acceptable latency that an Endpoint can withstand in the transition from the L1 to L0 state. It is an indirect measure of the Endpoint's internal buffering. It sets the read-only value of the Endpoint L1 acceptable latency field of the `Device Capabilities` register. The Arria V Hard IP for PCI Express does not support the L0s or L1 states. However, in a switched system there may be links connected to switches that have L0s and L1 enabled. This parameter is set to allow system configuration software to read the acceptable latencies for all devices in the system and the exit latencies for each link to determine which links can enable Active State Power Management (ASPM). This setting is disabled for Root Ports. The default value of this parameter is 1 .µs. This is the safest setting for most designs.

Parameters Defined Separately for All Port Functions

The **Port Functions** tab allows you to specify parameter settings for up to eight functions. Each function has separate settings for the following parameters:

- **Base Address Registers for Function <n>**
- **Base and Limit Registers for Root Port Func <n>**
- **Device ID Registers for Function <n>**
- **PCI Express/PCI Capabilities for Func <n>**

When you click on a **Func<*n*>** tab under the **Port Functions** heading, the parameter settings automatically relate to the function currently selected.

Base Address Registers for Function <*n*>

Table 3–7 describes the Base Address (BAR) register parameters.

Table 3–7. Func0–Func7 BARs and Expansion ROM

Parameter	Value	Description
Type **0x010**, **0x014**, **0x018**, **0x01C**, **0x020**, **0x024**	**Disabled** **64-bit prefetchable memory** **32-bit non-prefetchable memory** **32-bit prefetchable memory** **I/O address space**	If you select 64-bit prefetchable memory, 2 contiguous BARs are combined to form a 64-bit prefetchable BAR; you must set the higher numbered BAR to **Disabled**. A non-prefetchable 64-bit BAR is not supported because in a typical system, the Root Port Type 1 Configuration Space sets the maximum non-prefetchable memory window to 32-bits. The BARs can also be configured as separate 32-bit prefetchable or non-prefetchable memories. The **I/O address space** BAR is only available for the **Legacy Endpoint**.
Size	**16 Bytes–8 EBytes**	The **Endpoint** and **Root Port** variants support the following memory sizes: ■ ×1, ×4: 128 bytes–2 GBytes or 8 EBytes ■ ×8: 4 KBytes–2 GBytes or 8 EBytes (2 GBytes for 32-bit addressing and 8 EBytes for 64-bit addressing) The **Legacy Endpoint** supports the following I/O space BARs: ■ ×1, ×4:16 bytes–4 KBytes ■ ×8: 4 KBytes
Expansion ROM		
Size	**Disabled** **4 KBytes–16 MBytes**	Specifies the size of the optional ROM.

Base and Limit Registers for Root Port Func <*n*>

If you specify a Root Port for function 0, the settings for **Base and Limit Registers** required by Root Ports appear after the **Base Address Register** heading. These settings are stored in the Type 1 Configuration Space for Root Ports. They are used for TLP routing and specify the address ranges assigned to components that are downstream of the Root Port or bridge. Function 0 is the only function that provides the Root Port option for **Port type**.

For more information, refer to the *PCI-to-PCI Bridge Architecture Specification*.

Table 3–8 describes the Base and Limit registers parameters.

Table 3–8. Base and Limit Registers

Parameter	Value	Description
Input/Output	**Disable** **16-bit I/O addressing** **32-bit I/O addressing**	Specifies the address widths for the `IO base` and `IO limit` registers.
Prefetchable memory	**Disable** **32-bit I/O addressing** **64-bit I/O addressing**	Specifies the address widths for the `Prefetchable Memory Base` register and `Prefetchable Memory Limit` register.

Device ID Registers for Function *<n>*

Table 3–9 lists the default values of the read-only Device ID registers. You can use the parameter editor to change the values of these registers. At run time, you can change the values of these registers using the reconfiguration block signals. For more information, refer to "Reconfiguration Block Signals" on page 6–41.

Table 3–9. Device ID Registers for Function *<n>*

Register Name/ Offset Address	Range	Default Value	Description
Vendor ID **0x000**	16 bits	0x00000000	Sets the read-only value of the `Vendor ID` register. This parameter can not be set to 0xFFFF per the PCI Express Specification.
Device ID 0x000	16 bits	0x00000001	Sets the read-only value of the `Device ID` register.
Revision ID 0x008	8 bits	0x00000001	Sets the read-only value of the `Revision ID` register.
Class code **0x008**	24 bits	0x00000000	Sets the read-only value of the `Class Code` register.
Subsystem Vendor ID 0x02C	16 bits	0x00000000	Sets the read-only value of the `Subsystem Vendor ID` register. This parameter cannot be set to 0xFFFF per the *PCI Express Base Specification 2.1*. This register is available only for Endpoint designs which require the use of the Type 0 PCI Configuration register.
Subsystem Device ID **0x02C**	16 bits	0x0000000	Sets the read-only value of the `Subsystem Device ID` register. This register is only available for Endpoint designs, which require the use of the Type 0 PCI Configuration Space.

PCI Express/PCI Capabilities for Func *<n>*

The following sections describe the PCI Express and PCI Capabilities for each function.

Device

Table 3–10 describes the Link Capabilities register parameters.

Table 3–10. Function Level Reset

Parameter	Value	Description
Function level reset	**On/Off**	Turn **On** this option to provide separate reset for this function.

Link

Table 3–12 describes the Link Capabilities register parameters.

Table 3–11. Link 0x090

Parameter	Value	Description
Data link layer active reporting	**On/Off**	Turn **On** this option for a downstream port, if the component supports the optional capability of reporting the DL_Active state of the Data Link Control and Management State Machine. For a hot-plug capable downstream port (as indicated by the `Hot-Plug Capable` field of the `Slot Capabilities` register), this option must be turned **On**. For upstream ports and components that do not support this optional capability, turn **Off** this option. This parameter is only supported in Root Port mode.
Surprise down reporting	**On/Off**	When this option is **On**, a downstream port supports the optional capability of detecting and reporting the surprise down error condition. This parameter is only supported in Root Port mode.

MSI

Table 3–12 describes the MSI Capabilities register parameters.

Table 3–12. MSI and MSI-X Capabilities 0x050–0x05C,

Parameter	Value	Description
MSI messages requested	**1, 2, 4, 8, 16**	Specifies the number of messages the Application Layer can request. Sets the value of the `Multiple Message Capable` field of the `Message Control` register, 0x050[31:16].

MSI-X

Table 3–12 describes the MSI-X Capabilities register parameters.

Table 3–13. MSI and MSI-X Capabilities 0x068–0x06C

Parameter	Value	Description
Implement MSI-X	**On/Off**	When **On**, enables the MSI-X functionality.
Table size 0x068[26:16]	[10:0]	System software reads this field to determine the MSI-X Table size <*n*>, which is encoded as <*n*–1>. For example, a returned value of 2047 indicates a table size of 2048. This field is read-only. Legal range is 0–2047 (2^{11}).
Table Offset	[31:3]	Points to the base of the MSI-X Table. The lower 3 bits of the table BAR indicator (BIR) are set to zero by software to form a 32-bit qword-aligned offset. This field is read-only. Legal range is 0–2^{28}.
Table BAR Indicator	<5–1>:0	Specifies which one of a function's BARs, located beginning at 0x10 in Configuration Space, is used to map the MSI-X table into memory space. This field is read-only. Legal range is 0–5.
Pending Bit Array (PBA) Offset	31:3	Used as an offset from the address contained in one of the function's Base Address registers to point to the base of the MSI-X PBA. The lower 3 bits of the PBA BIR are set to zero by software to form a 32-bit qword-aligned offset. This field is read-only. Legal range is 0–2^{28}.
PBA BAR Indicator (BIR)	<5–1>:0	Indicates which of a function's Base Address registers, located beginning at 0x10 in Configuration Space, is used to map the function's MSI-X PBA into memory space. This field is read-only. Legal range is 0–5.

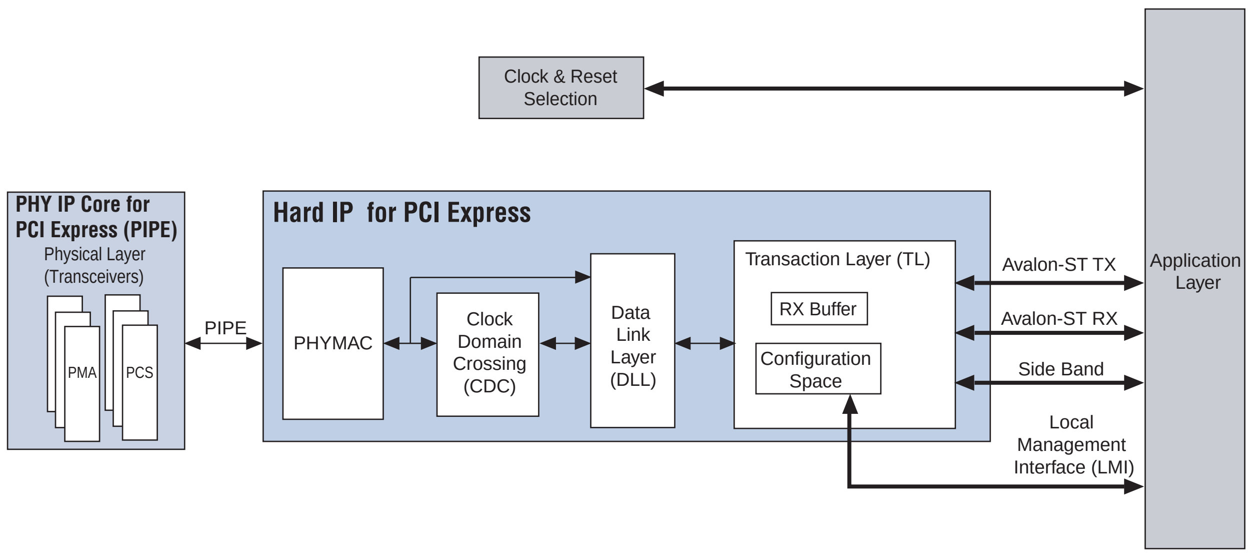
4. IP Core Architecture

This chapter describes the architecture of the Arria V Hard IP for PCI Express. The Arria V Hard IP for PCI Express implements the complete PCI Express protocol stack as defined in the *PCI Express Base Specification 2.1*. The protocol stack includes the following layers:

- *Transaction Layer*—The Transaction Layer contains the Configuration Space, the RX and TX channels, the RX buffer, and flow control credits.
- *Data Link Layer*—The Data Link Layer, located between the Physical Layer and the Transaction Layer, manages packet transmission and maintains data integrity at the link level. Specifically, the Data Link Layer performs the following tasks:
  - Manages transmission and reception of Data Link Layer Packets (DLLPs)
  - Generates all transmission cyclical redundancy code (CRC) values and checks all CRCs during reception
  - Manages the retry buffer and retry mechanism according to received ACK/NAK Data Link Layer packets
  - Initializes the flow control mechanism for DLLPs and routes flow control credits to and from the Transaction Layer
- *Physical Layer*—The Physical Layer initializes the speed, lane numbering, and lane width of the PCI Express link according to packets received from the link and directives received from higher layers.

Figure 4–2 provides a high-level block diagram of the Arria V Hard IP for PCI Express.

Figure 4–1. Arria V Hard IP for PCI Express with Avalon-ST Interface



As Figure 4–1 illustrates, an Avalon-ST interface provides access to the Application Layer which can be either 64 or 128 bits. Table 4–1 provides the Application Layer clock frequencies.

Table 4–1. Application Layer Clock Frequencies

Lanes	Gen1	Gen2
×1	125 MHz @ 64 bits or 62.5 MHz @ 64 bits	125 MHz @ 64 bits
×4	125 MHz @ 64 bits	125 MHz @ 128 bits
×8	125 MHz @ 128 bits	—

The following interfaces provide access to the Application Layer's Configuration Space Registers:

- The LMI interface
- For Root Ports, you can also access the Configuration Space Registers with a Configuration Type TLP using the Avalon-ST interface. A Type 0 Configuration TLP is used to access the Root Port Configuration Space Registers, and a Type 1 Configuration TLP is used to access the Configuration Space Registers of downstream components, typically Endpoints on the other side of the link.

The Hard IP includes dedicated clock domain crossing logic (CDC) between the PHYMAC and Data Link Layers.

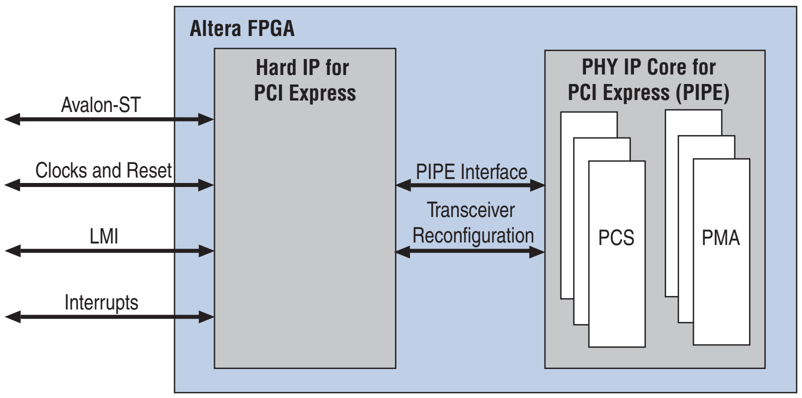
This chapter provides an overview of the architecture of the Arria V Hard IP for PCI Express. It includes the following sections:

- Key Interfaces
- Protocol Layers
- Multi-Function Support

Key Interfaces

The following sections introduce the functionality of the interfaces shown in Figure 4–2.

Figure 4–2. Block Diagram



Avalon-ST Interface

An Avalon-ST interface connects the Application Layer and the Transaction Layer. This is a point-to-point, streaming interface designed for high throughput applications. The Avalon-ST interface includes the RX and TX datapaths.

For more information about the Avalon-ST interface, including timing diagrams, refer to the *Avalon Interface Specifications*.

RX Datapath

The RX datapath transports data from the Transaction Layer to the Application Layer's Avalon-ST interface. Masking of non-posted requests is partially supported. Refer to the description of the `rx_st_mask` signal for further information about masking. For more detailed information about the RX datapath, refer to "Avalon-ST RX Interface" on page 5–3.

TX Datapath

The TX datapath transports data from the Application Layer's Avalon-ST interface to the Transaction Layer. The Hard IP provides credit information to the Application Layer for posted headers, posted data, non-posted headers, non-posted data, completion headers and completion data.

The Application Layer may track credits consumed and use the credit limit information to calculate the number of credits available. However, to enforce the PCI Express Flow Control (FC) protocol, the Hard IP also checks the available credits before sending a request to the link, and if the Application Layer violates the available credits for a TLP it transmits, the Hard IP blocks that TLP and all future TLPs until credits become available. By tracking the credit consumed information and calculating the credits available, the Application Layer can optimize performance by selecting for transmission only the TLPs that have credits available. Refer to "Avalon-ST TX Interface" on page 5–12 for more information about the signals in this interface.

Clocks and Reset

The *PCI Express Base Specification* requires an input reference clock, which is called `refclk` in this design. Although the *PCI Express Base Specification* stipulates that the frequency of this clock be 100 MHz, the Hard IP also accepts a 125 MHz reference clock as a convenience. You can specify the frequency of your input reference clock using the parameter editor under the **System Settings** heading.

The *PCI Express Base Specification 2.1*, requires the following three reset types:

- *cold reset*—A hardware mechanism for setting or returning all port states to the initial conditions following the application of power.
- *warm reset*—A hardware mechanism for setting or returning all port states to the initial conditions without cycling the supplied power.
- *hot reset* —A reset propagated across a PCIe link using a Physical Layer mechanism.

The *PCI Express Base Specification* also requires a system configuration time of 100 ms. To meet this specification, the Arria V Hard IP for PCI Express includes an embedded hard reset controller. For more information about clocks and reset, refer to the "Clock Signals" on page 5–20 and "Reset Signals" on page 5–20.

Local Management Interface (LMI Interface)

The LMI bus provides access to the PCI Express Configuration Space in the Transaction Layer. For more LMI details, refer to "LMI Signals" on page 5–33.

Interrupts

The Arria V Hard IP for PCI Express offers three interrupt mechanisms:

- Message Signaled Interrupts (MSI)— MSI uses the Transaction Layer's request-acknowledge handshaking protocol to implement interrupts. The MSI Capability structure is stored in the Configuration Space and is programmable using Configuration Space accesses.
- MSI-X—The Transaction Layer generates MSI-X messages which are single dword memory writes. In contrast to the MSI capability structure, which contains all of the control and status information for the interrupt vectors, the MSI-X Capability structure points to an MSI-X table structure and MSI-X PBA structure which are stored in memory.

- Legacy interrupts—The `app_int_sts` input port controls legacy interrupt generation. When `app_int_sts` is asserted, the Hard IP generates an Assert_INT<*n*> message TLP. For more detailed information about interrupts, refer to "Interrupt Signals for Endpoints" on page 5–23.

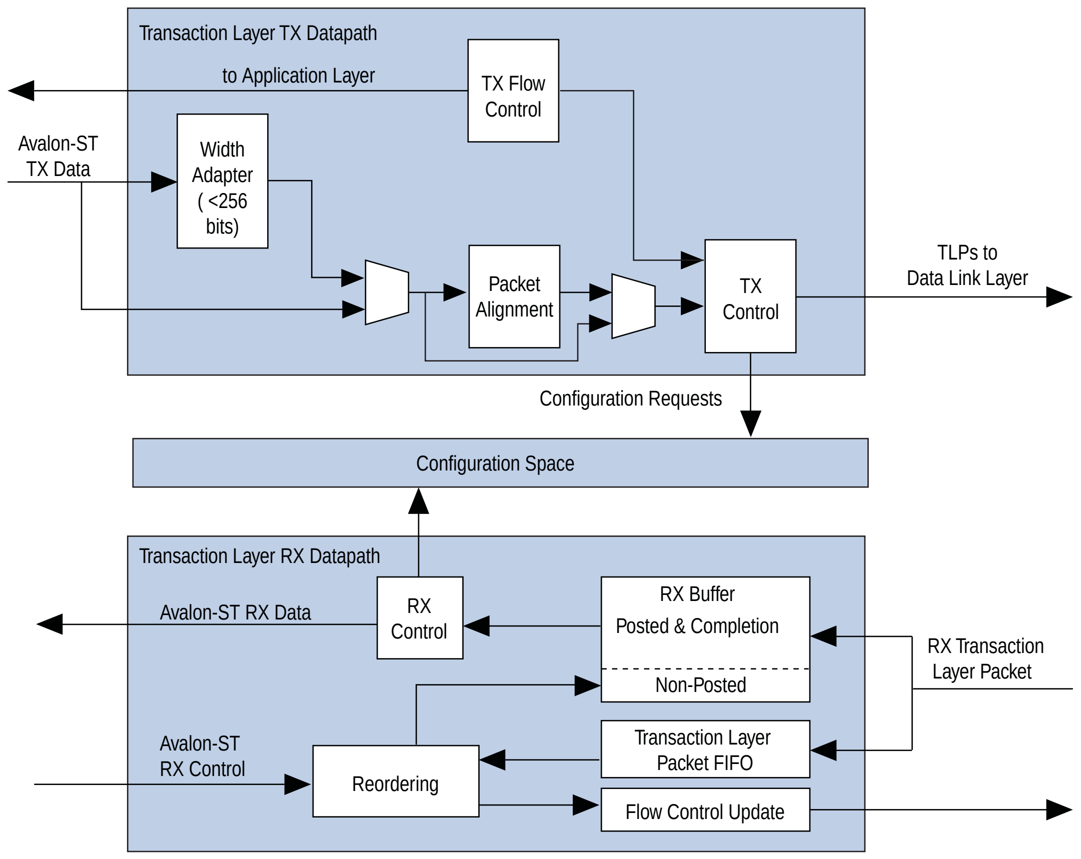
Protocol Layers

This section describes the Transaction Layer, Data Link Layer, and Physical Layer in more detail.

Transaction Layer

The Transaction Layer is located between the Application Layer and the Data Link Layer. It generates and receives Transaction Layer Packets. Figure 4–3 illustrates the Transaction Layer. As Figure 4–3 illustrates, the Transaction Layer includes three sub-blocks: the TX datapath, the Configuration Space, and the RX datapath.

Figure 4–3. Architecture of the Transaction Layer: Dedicated Receive Buffer



Tracing a transaction through the RX datapath includes the following steps:

1. The Transaction Layer receives a TLP from the Data Link Layer.
2. The Transaction Layer determines whether the TLP is well formed and directs the packet based on traffic class (TC).

3. TLPs are stored in a specific part of the RX buffer depending on the type of transaction (posted, non-posted, and completion).
4. The TLP FIFO block stores the address of the buffered TLP.
5. The receive reordering block reorders the queue of TLPs as needed, fetches the address of the highest priority TLP from the TLP FIFO block, and initiates the transfer of the TLP to the Application Layer.
6. When ECRC generation and forwarding are enabled, the Transaction Layer forwards the ECRC dword to the Application Layer.

Tracing a transaction through the TX datapath involves the following steps:

1. The Transaction Layer informs the Application Layer that sufficient flow control credits exist for a particular type of transaction using the TX credit signals. The Application Layer may choose to ignore this information.
2. The Application Layer requests permission to transmit a TLP. The Application Layer must provide the transaction and must be prepared to provide the entire data payload in consecutive cycles.
3. The Transaction Layer verifies that sufficient flow control credits exist and acknowledges or postpones the request.
4. The Transaction Layer forwards the TLP to the Data Link Layer.

Configuration Space

The Configuration Space implements the following Configuration Space Registers and associated functions:

- Header Type 0 Configuration Space for Endpoints
- Header Type 1 Configuration Space for Root Ports
- PCI Power Management Capability Structure
- Message Signaled Interrupt (MSI) Capability Structure
- Message Signaled Interrupt–X (MSI–X) Capability Structure
- PCI Express Capability Structure
- Advanced Error Reporting (AER) Capability Structure

The Configuration Space also generates all messages (PME#, INT, Error, Slot Power Limit), MSI requests, and completion packets from configuration requests that flow in the direction of the root complex, except Slot Power Limit messages, which are generated by a downstream port. All such transactions are dependent upon the content of the PCI Express Configuration Space as described in the *PCI Express Base Specification Revision 2.1*.

Refer To "Configuration Space Register Content" on page 6–1 or Chapter 7 in the *PCI Express Base Specification 2.1* for the complete content of these registers.

Data Link Layer

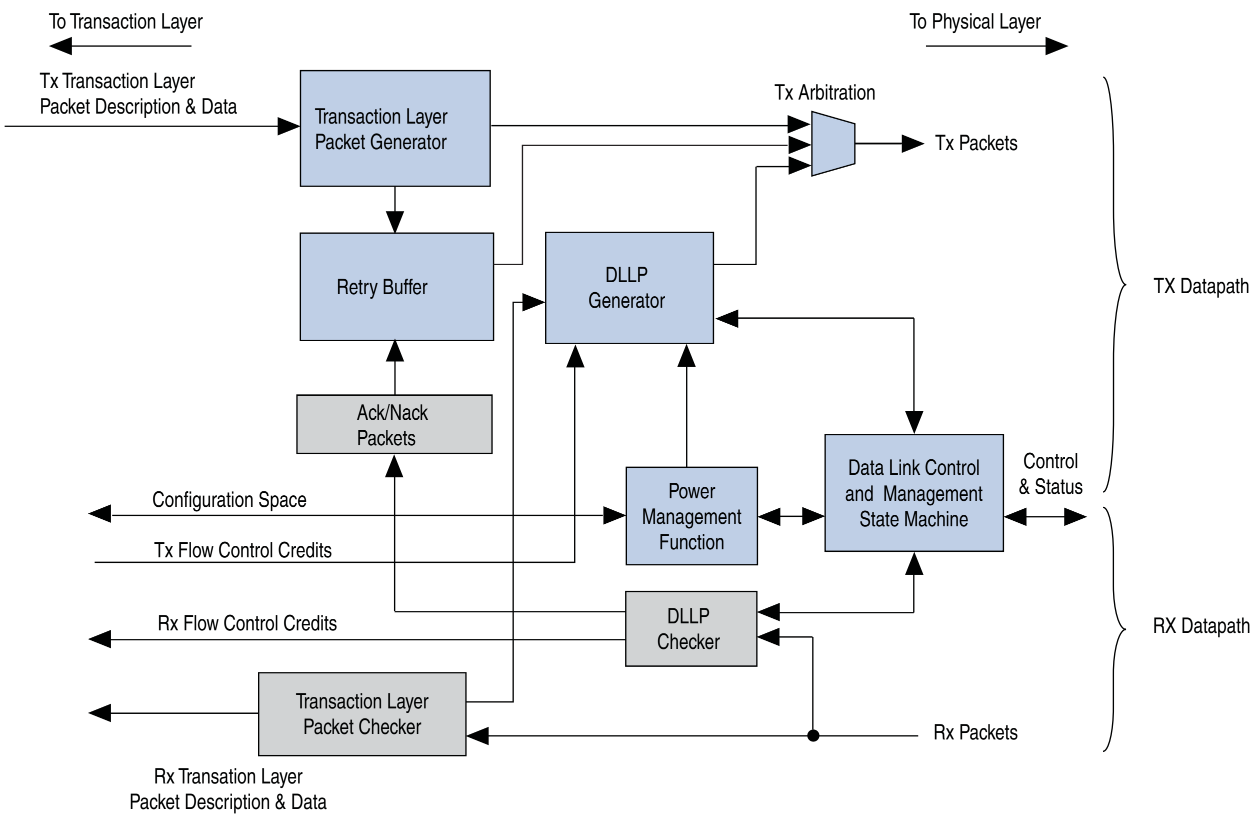
The Data Link Layer (DLL) is located between the Transaction Layer and the Physical Layer. It maintains packet integrity and for communicates (by DLL packet transmission) at the PCI Express link level (as opposed to component communication by TLP transmission in the interconnect fabric).

The DLL implements the following functions:

- Link management through the reception and transmission of DLL packets (DLLP), which are used for the following functions:
  - For power management of DLLP reception and transmission
  - To transmit and receive `ACK/NACK` packets
- Data integrity through generation and checking of CRCs for TLPs and DLLPs
- TLP retransmission in case of `NAK` DLLP reception using the retry buffer
- Management of the retry buffer
- Link retraining requests in case of error through the Link Training and Status State Machine (LTSSM) of the Physical Layer

Figure 4–4 illustrates the architecture of the DLL.

Figure 4–4. Data Link Layer


To Transaction Layer
To Physical Layer
Tx Transaction Layer Packet Description & Data
Transaction Layer Packet Generator
Tx Arbitration
Tx Packets
Retry Buffer
DLLP Generator
TX Datapath
Ack/Nack Packets
Configuration Space
Tx Flow Control Credits
Power Management Function
Data Link Control and Management State Machine
Control & Status
Rx Flow Control Credits
DLLP Checker
RX Datapath
Transaction Layer Packet Checker
Rx Packets
Rx Transation Layer Packet Description & Data

The DLL has the following sub-blocks:

- Data Link Control and Management State Machine—This state machine is synchronized with the Physical Layer's LTSSM state machine and also connects to the Configuration Space Registers. It initializes the link and flow control credits and reports status to the Configuration Space.
- Data Link Layer Packet Generator and Checker—This block is associated with the DLLP's 16-bit CRC and maintains the integrity of transmitted packets.
- Transaction Layer Packet Generator—This block generates transmit packets, generating a sequence number and a 32-bit CRC (LCRC). The packets are also sent to the retry buffer for internal storage. In retry mode, the TLP generator receives the packets from the retry buffer and generates the CRC for the transmit packet.
- Retry Buffer—The retry buffer stores TLPs and retransmits all unacknowledged packets in the case of NAK DLLP reception. For ACK DLLP reception, the retry buffer discards all acknowledged packets.
- ACK/NAK Packets—The ACK/NAK block handles ACK/NAK DLLPs and generates the sequence number of transmitted packets.
- Transaction Layer Packet Checker—This block checks the integrity of the received TLP and generates a request for transmission of an ACK/NAK DLLP.
- TX Arbitration—This block arbitrates transactions, prioritizing in the following order:

  a. Initialize FC Data Link Layer packet

  b. ACK/NAK DLLP (high priority)

  c. Update FC DLLP (high priority)

  d. PM DLLP

  e. Retry buffer TLP

  f. TLP

  g. Update FC DLLP (low priority)

  h. ACK/NAK FC DLLP (low priority)

Physical Layer

The Physical Layer is the lowest level of the Arria V Hard IP for PCI Express. It is the layer closest to the link. It encodes and transmits packets across a link and accepts and decodes received packets. The Physical Layer connects to the link through a high-speed SERDES interface running at 2.5 Gbps for Gen1 implementations and at 2.5 or 5.0 Gbps for Gen2 implementations.

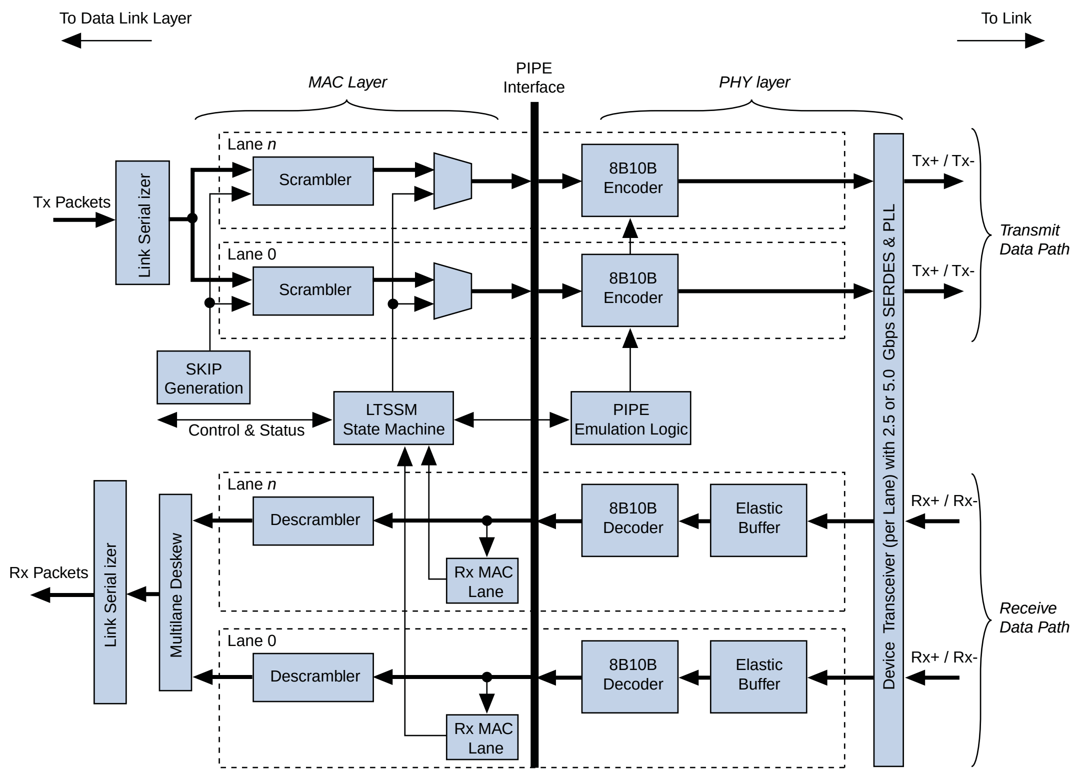
The Physical Layer is responsible for the following actions:

- Initializing the link
- Scrambling/descrambling and 8B/10B encoding/decoding of 2.5 Gbps (Gen1) or 5.0 Gbps (Gen2) per lane 8B/10B
- Serializing and deserializing data
- Operating the PIPE 2.0 Interface

- Implementing auto speed negotiation
- Transmitting and decoding the training sequence
- Providing hardware autonomous speed control
- Implementing auto lane reversal

Figure 4–5 illustrates the Physical Layer architecture.

Figure 4–5. Physical Layer



The Physical Layer is subdivided by the PIPE Interface Specification into two layers (bracketed horizontally in Figure 4–5):

- Media Access Controller (MAC) Layer—The MAC layer includes the LTSSM and the scrambling/descrambling and multilane deskew functions.
- PHY Layer—The PHY layer includes the 8B/10B encode/decode functions, elastic buffering, and serialization/deserialization functions.

The Physical Layer integrates both digital and analog elements. Intel designed the PIPE interface to separate the MAC from the PHY. The Arria V Hard IP for PCI Express complies with the PIPE interface specification.

The PHYMAC block is divided in four main sub-blocks:

- MAC Lane—Both the RX and the TX path use this block.
  - On the RX side, the block decodes the Physical Layer Packet and reports to the LTSSM the type and number of TS1/TS2 ordered sets received.
  - On the TX side, the block multiplexes data from the DLL and the LTSTX sub-block. It also adds lane specific information, including the lane number and the force PAD value when the LTSSM disables the lane during initialization.
- LTSSM—This block implements the LTSSM and logic that tracks what is received and transmitted on each lane.
  - For transmission, it interacts with each MAC lane sub-block and with the LTSTX sub-block by asserting both global and per-lane control bits to generate specific Physical Layer packets.
  - On the receive path, it receives the Physical Layer Packets reported by each MAC lane sub-block. It also enables the multilane deskew block and the delay required before the TX alignment sub-block can move to the recovery or low power state. A higher layer can direct this block to move to the recovery, disable, hot reset or low power states through a simple request/acknowledge protocol. This block reports the Physical Layer status to higher layers.
- LTSTX (Ordered Set and SKP Generation)—This sub-block generates the Physical Layer Packet. It receives control signals from the LTSSM block and generates Physical Layer Packet for each lane. It generates the same Physical Layer Packet for all lanes and PAD symbols for the link or lane number in the corresponding TS1/TS2 fields.

  The block also handles the receiver detection operation to the PCS sub-layer by asserting predefined PIPE signals and waiting for the result. It also generates a SKP Ordered Set at every predefined timeslot and interacts with the TX alignment block to prevent the insertion of a SKP Ordered Set in the middle of packet.
- Deskew—This sub-block performs the multilane deskew function and the RX alignment between the number of initialized lanes and the 64-bit data path.

  The multilane deskew implements an eight-word FIFO for each lane to store symbols. Each symbol includes eight data bits, one disparity bit, and one control bit. The FIFO discards the FTS, COM, and SKP symbols and replaces PAD and IDL with D0.0 data. When all eight FIFOs contain data, a read can occur.

  When the multilane lane deskew block is first enabled, each FIFO begins writing after the first COM is detected. If all lanes have not detected a COM symbol after seven clock cycles, they are reset and the resynchronization process restarts, or else the RX alignment function recreates a 64-bit data word which is sent to the DLL.

Multi-Function Support

The Arria V Hard IP for PCI Express supports up to eight functions for Endpoints. You set up the each function under the Port Functions heading in the parameter editor. You can configure Arria V devices to include both Native and Legacy Endpoints. Each function replicates the Configuration Space Registers, including logic for Tag Tracking and Error detection.

Because the Configuration Space is replicated for each function, some Configuration Space Register settings may conflict. Arbitration logic resolves differences when settings contain different values across multiple functions. The arbitration logic implements the rules for resolving conflicts as specified in the *PCI Express Base Specification 2.0*. Examples of settings that require arbitration include the following features:

- Link Control settings
- ECRC generation and checking
- Error detection and logging for non-function-specific errors
- Error message collapsing
- Maximum payload size (All functions use the largest specified maximum payload setting.)
- Interrupt message collapsing

You can access the Configuration Space Registers for the active function using the LMI interface. In Root Port mode, you can also access the Configuration Space Registers using a Configuration Type TLP. Refer to "Configuration Space Register Content" on page 6–1 for more information about the Configuration Space Registers.

5. IP Core Interfaces

This chapter describes the signals that are part of the Arria V Hard IP for PCI Express IP core. Figure 5–1 on page 5–2 illustrates the top-level signals IP core.

Because the Arria V Hard IP for PCI Express offers exactly the same feature set in the MegaWizard Plug-In Manager and Qsys design flows, your decision about which design flow to use depends on whether you want to integrate the Arria V Hard IP for PCI Express using RTL instantiation or Qsys. The Qsys system integration tool automatically generates the interconnect logic between the IP components in your system, saving time and effort. Refer to "MegaWizard Plug-In Manager Design Flow" on page 2–3 and "Qsys Design Flow" on page 2–9 for a description of the steps involved in the two design flows.

Table 5–1 lists each interface and provides a link to the subsequent sections that describe each signal. The signals are described in the order in which they are shown in Figure 5–1.

Table 5–1. Signal Groups in the Arria V Hard IP for PCI Express

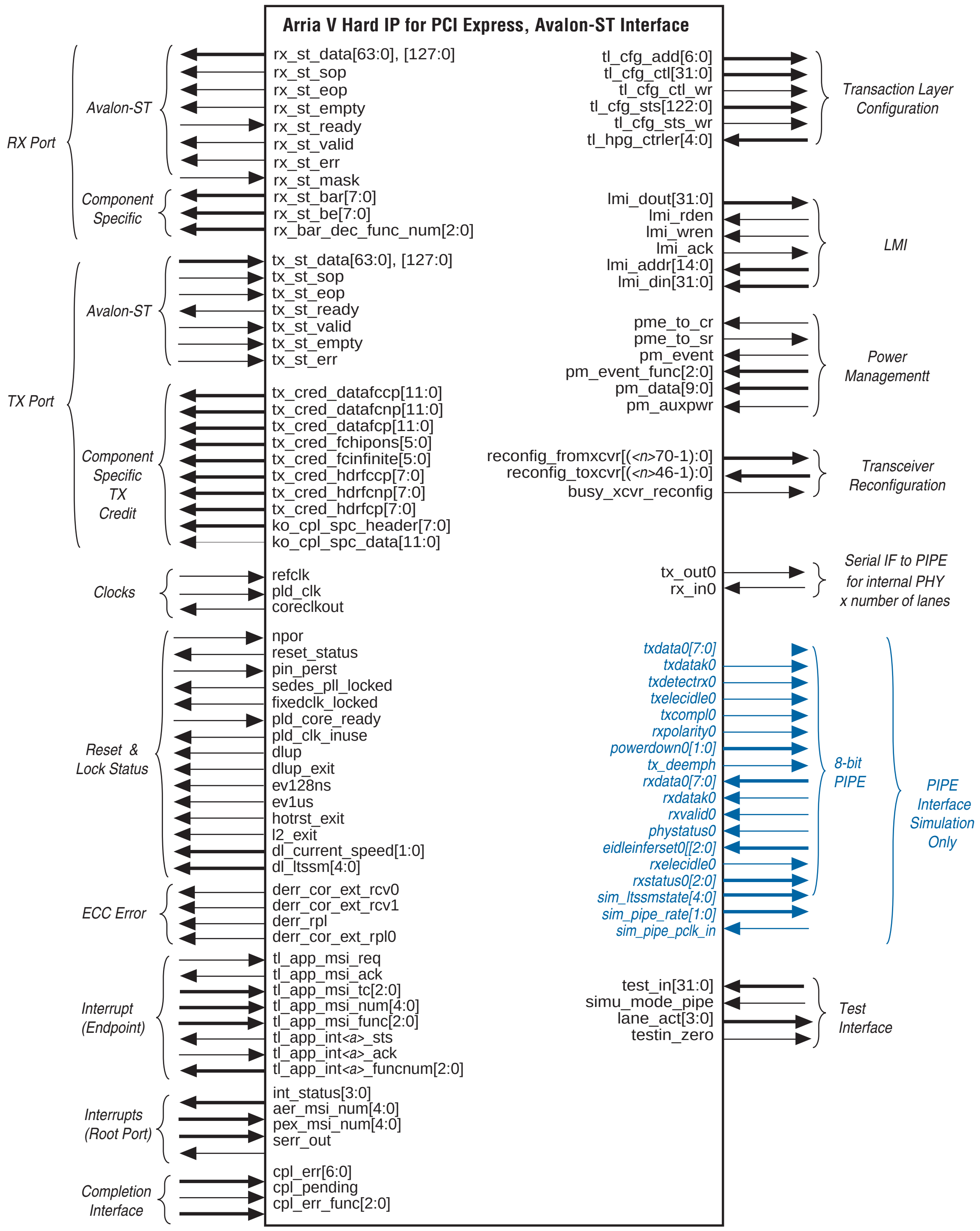
Signal Group	Description
Logical	
Avalon-ST RX	"Avalon-ST RX Interface" on page 5–3
Avalon-ST TX	"Avalon-ST TX Interface" on page 5–12
Clock	"Clock Signals" on page 5–20
Reset and link training	"Reset Signals" on page 5–20
ECC error	"ECC Error Signals" on page 5–23
Interrupt	"Interrupts for Endpoints" on page 5–23
Interrupt and global error	"Interrupts for Root Ports" on page 5–24
Configuration space	"Transaction Layer Configuration Space Signals" on page 5–26
LMI	"LMI Signals" on page 5–33
Completion	"Completion Side Band Signals" on page 5–25
Power management	"Power Management Signals" on page 5–36
Physical	
Transceiver control	"Transceiver Reconfiguration" on page 5–38
Serial	"Serial Interface Signals" on page 5–38
PIPE (1)	"PIPE Interface Signals" on page 5–41
Test	
Test	"Test Signals" on page 5–43

Note to Table 5–1:

(1) Provided for simulation only

Figure 5–1 illustrates the top-level signals in Arria V Hard IP for PCI Express IP core. Signal names that include <*a*> also exist for functions 1 to 7.

Figure 5–1. Signals in the Arria V Hard IP for PCI Express with Avalon-ST Interface



Avalon-ST RX Interface

Table 5–2 describes the signals that comprise the Avalon-ST RX Datapath. The RX data signal can be 64 or 128 bits.

Table 5–2. 64- or 128-Bit Avalon-ST RX Datapath (Part 1 of 2)

Signal	Width	Dir	Avalon-ST Type	Description
`rx_st_data`	64 128	O	`data`	Receive data bus. Refer to the figures below for the mapping of the Transaction Layer's TLP information to `rx_st_data` and examples of the timing of this interface. Note that the position of the first payload dword depends on whether the TLP address is qword aligned. The mapping of message TLPs is the same as the mapping of TLPs with 4 dword headers. When using a 64-bit Avalon-ST bus, the width of `rx_st_data` is 64. When using a 128-bit Avalon-ST bus, the width of `rx_st_data` is 128.
`rx_st_sop`	1	O	`start of packet`	Indicates that this is the first cycle of the TLP when `rx_st_valid` is asserted.
`rx_st_eop`	1	O	`end of packet`	Indicates that this is the last cycle of the TLP when `rx_st_valid` is asserted.
`rx_st_empty`	1	O	`empty`	Indicates the number of empty qwords in `rx_st_data`. Not used when `rx_st_data` is 64 bits. When asserted, indicates that the upper qword contains is empty, *does not contain valid data*.
`rx_st_ready`	1	I	`ready`	Indicates that the Application Layer is ready to accept data. The Application Layer deasserts this signal to throttle the data stream. If `rx_st_ready` is asserted by the Application Layer on cycle *<n>*, then *<n +* `readyLatency`*>* is a ready cycle, during which the Transaction Layer may assert `valid` and transfer data. The RX interface supports a `readyLatency` of 2 cycles.
`rx_st_valid`	1	O	`valid`	Clocks `rx_st_data` into the Application Layer. Deasserts within 2 clocks of `rx_st_ready` deassertion and reasserts within 2 clocks of `rx_st_ready` the assertion if more data is available to send.
`rx_st_err`	1	O	`error`	Indicates that there is an uncorrectable ECC error in the internal RX buffer. Active when ECC is enabled. ECC is automatically enabled by the Quartus II assembler. ECC corrects single-bit errors and detects double-bit errors on a per byte basis. When an uncorrectable ECC error is detected, `rx_st_err` is asserted for at least 1 cycle while `rx_st_valid` is asserted. Altera recommends resetting the Arria V Hard IP for PCI Express IP core when an uncorrectable (double-bit) ECC error is detected.
Component Specific Signals				
`rx_st_mask`	1	I	`component specific`	The Application Layer asserts this signal to tell the Hard IP to stop sending non-posted requests. This signal can be asserted at any time. The total number of non-posted requests that can be transferred to the Application Layer after `rx_st_mask` is asserted is not greater than 10.

Table 5–2. 64- or 128-Bit Avalon-ST RX Datapath (Part 2 of 2)

Signal	Width	Dir	Avalon-ST Type	Description
`rx_st_bar`	8	O	`component specific`	The decoded BAR bits for the TLP. Valid for `MRd`, `MWr`, `IOWR`, and `IORD` TLPs; ignored for the completion or message TLPs. Valid during the cycle in which `rx_st_sop` is asserted. Figure 5–6 illustrates the timing of this signal for 64-bit data. Figure 5–9 illustrates the timing of this signal for 128-bit data. The following encodings are defined for Endpoints: ■ Bit 0: BAR 0 ■ Bit 1: BAR 1 ■ Bit 2: Bar 2 ■ Bit 3: Bar 3 ■ Bit 4: Bar 4 ■ Bit 5: Bar 5 ■ Bit 6: Expansion ROM ■ Bit 7: Reserved The following encodings are defined for Root Ports: ■ Bit 0: BAR 0 ■ Bit 1: BAR 1 ■ Bit 2: Primary Bus number ■ Bit 3: Secondary Bus number ■ Bit 4: Secondary Bus number to Subordinate Bus number window ■ Bit 5: I/O window ■ Bit 6: Non-Prefetchable window ■ Bit 7: Prefetchable window

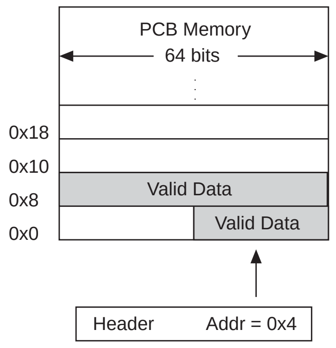
`rx_st_be`	8	O	`component specific`	Byte enables corresponding to the `rx_st_data`. The byte enable signals only apply to PCI Express TLP payload fields. When using 64-bit Avalon-ST bus, the width of `rx_st_be` is 8 bits. This signal is optional. You can derive the same information by decoding the `FBE` and `LBE` fields in the TLP header. The byte enable bits correspond to data bytes as follows:
				`rx_st_data[63:56] = rx_st_be[7]`
				`rx_st_data[55:48] = rx_st_be[6]`
				`rx_st_data[47:40] = rx_st_be[5]`
				`rx_st_data[39:32] = rx_st_be[4]`
				`rx_st_data[31:24] = rx_st_be[3]`
				`rx_st_data[23:16] = rx_st_be[2]`
				`rx_st_data[15:8] = rx_st_be[1]`
				`rx_st_data[7:0] = rx_st_be[0]`
				This signal is deprecated.

`rx_bar_dec_func_num`	3	O	`component specific`	Specifies which function the `rx_st_bar` signal applies to.

For more information about the Avalon-ST protocol, refer to the *Avalon Interface Specifications*.

To facilitate the interface to 64-bit memories, the Arria V Hard IP for PCI Express aligns data to the qword or 64 bits by default; consequently, if the header presents an address that is not qword aligned, the Hard IP block shifts the data within the qword to achieve the correct alignment. Figure 5–2 shows how an address that is not qword aligned, 0x4, is stored in memory. The byte enables only qualify data that is being written. This means that the byte enables are undefined for 0x0–0x3. This example corresponds to Figure 5–3 on page 5–6. Qword alignment applies to all types of request TLPs with data, including memory writes, configuration writes, and I/O writes. The alignment of the request TLP depends on bit 2 of the request address. For completion TLPs with data, alignment depends on bit 2 of the `lower address` field. This bit is always 0 (aligned to qword boundary) for completion with data TLPs that are for configuration read or I/O read requests

Figure 5–2. Qword Alignment



Refer to Appendix A, Transaction Layer Packet (TLP) Header Formats for the formats of all TLPs.

Table 5–3 shows the byte ordering for header and data packets.

Table 5–3. Mapping Avalon-ST Packets to PCI Express TLPs

Packet	TLP
Header0	pcie_hdr_byte0, pcie_hdr _byte1, pcie_hdr _byte2, pcie_hdr _byte3
Header1	pcie_hdr _byte4, pcie_hdr _byte5, pcie_hdr byte6, pcie_hdr _byte7
Header2	pcie_hdr _byte8, pcie_hdr _byte9, pcie_hdr _byte10, pcie_hdr _byte11
Header3	pcie_hdr _byte12, pcie_hdr _byte13, header_byte14, pcie_hdr _byte15
Data0	pcie_data_byte3, pcie_data_byte2, pcie_data_byte1, pcie_data_byte0
Data1	pcie_data_byte7, pcie_data_byte6, pcie_data_byte5, pcie_data_byte4
Data2	pcie_data_byte11, pcie_data_byte10, pcie_data_byte9, pcie_data_byte8
Data<*n*>	pcie_data_byte<*4n+3*>, pcie_data_byte<*4n+2*>, pcie_data_byte<*4n+1*>, pcie_data_byte<*n*>

Data Alignment and Timing for the 64-Bit Avalon-ST RX Interface

Figure 5–3 illustrates the mapping of Avalon-ST RX packets to PCI Express TLPs for a three dword header with non-qword aligned addresses with a 64-bit bus. In this example, the byte address is unaligned and ends with 0x4, causing the first data to correspond to `rx_st_data[63:32]`.

The Avalon-ST protocol, as defined in *Avalon Interface Specifications*, is big endian, while the Hard IP for PCI Express packs symbols into words in little endian format. Consequently, you cannot use the standard data format adapters available in Qsys.

Figure 5–3. 64-Bit Avalon-ST rx_st_data<*n*> Cycle Definition for 3-Dword Header TLP with Non-Qword Aligned Address

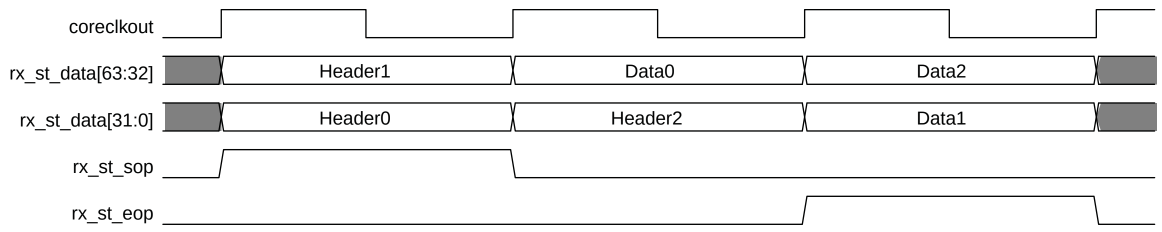


Figure 5–4 illustrates the mapping of Avalon-ST RX packets to PCI Express TLPs for a three dword header with qword aligned addresses. Note that the byte enables indicate the first byte of data is not valid and the last dword of data has a single valid byte.

Figure 5–4. 64-Bit Avalon-ST rx_st_data<n> Cycle Definition for 3-Dword Header TLP with Qword Aligned Address

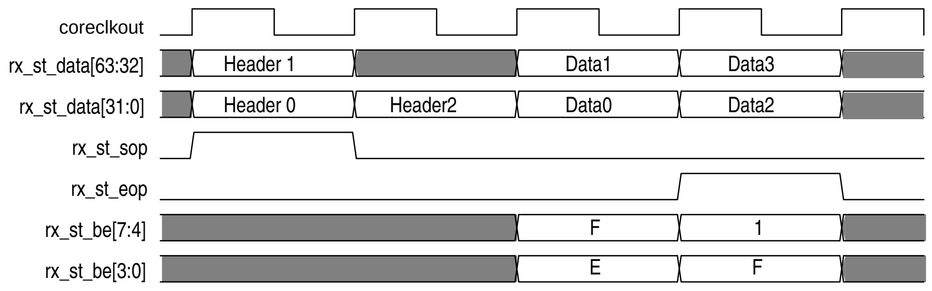


Figure 5–5 shows the mapping of Avalon-ST RX packets to PCI Express TLPs for TLPs for a four dword header with qword aligned addresses with a 64-bit bus.

Figure 5–5. 64-Bit Avalon-ST rx_st_data<*n*> Cycle Definitions for 4-Dword Header TLP with Qword Aligned Address

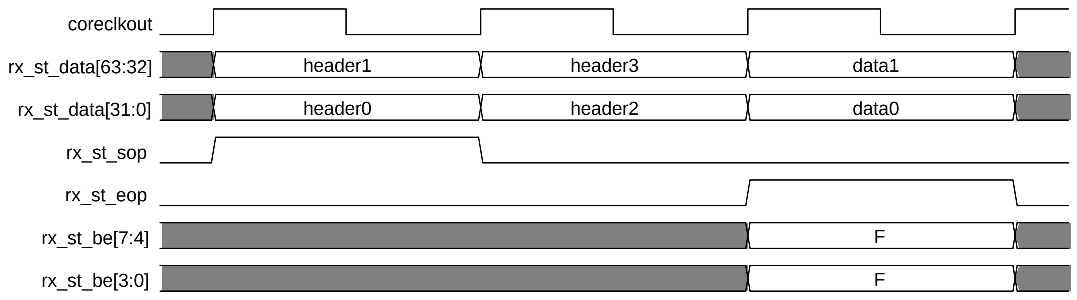
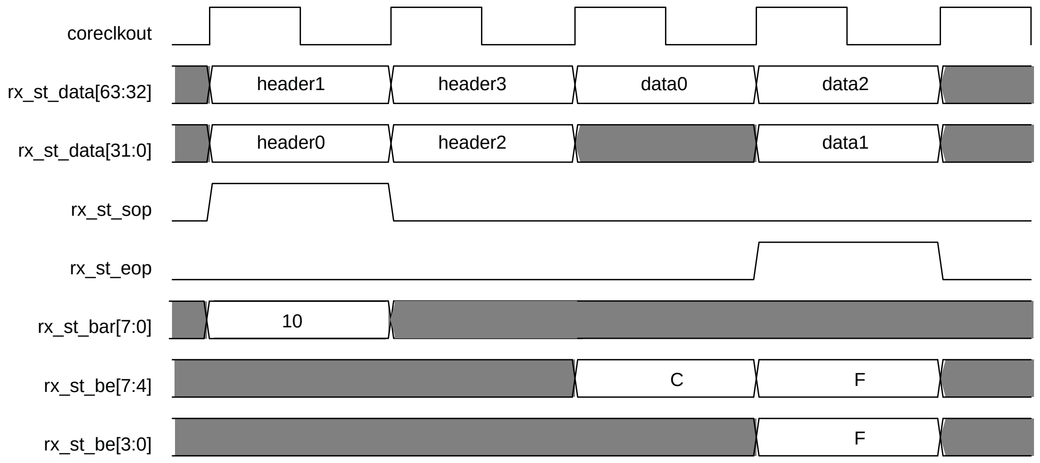


Figure 5–6 shows the mapping of Avalon-ST RX packet to PCI Express TLPs for TLPs for a four dword header with non-qword addresses with a 64-bit bus. Note that the address of the first dword is 0x4. The address of the first enabled byte is 0x6. This example shows one valid word in the first dword, as indicated by the `rx_st_be` signal.

Figure 5–6. 64-Bit Avalon-ST rx_st_data<*n*> Cycle Definitions for 4-Dword Header TLP with Non-Qword Address (1)



Note to Figure 5–6:

(1) `rx_st_be[7:4]` corresponds to `rx_st_data[63:32]`. `rx_st_be[3:0]` corresponds to `rx_st_data[31:0]`.

Figure 5–7 illustrates the timing of the RX interface when the Application Layer backpressures the Arria V Hard IP for PCI Express by deasserting `rx_st_ready`. The `rx_st_valid` signal must deassert within three cycles after `rx_st_ready` is deasserted. In this example, `rx_st_valid` is deasserted in the next cycle. `rx_st_data` is held until the Application Layer is able to accept it.

Figure 5–7. 64-Bit Application Layer Backpressures Transaction Layer

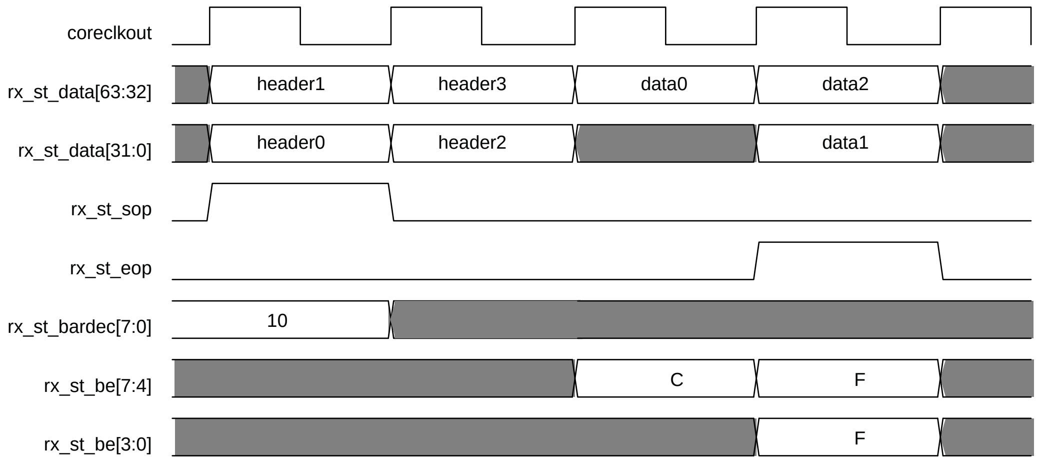
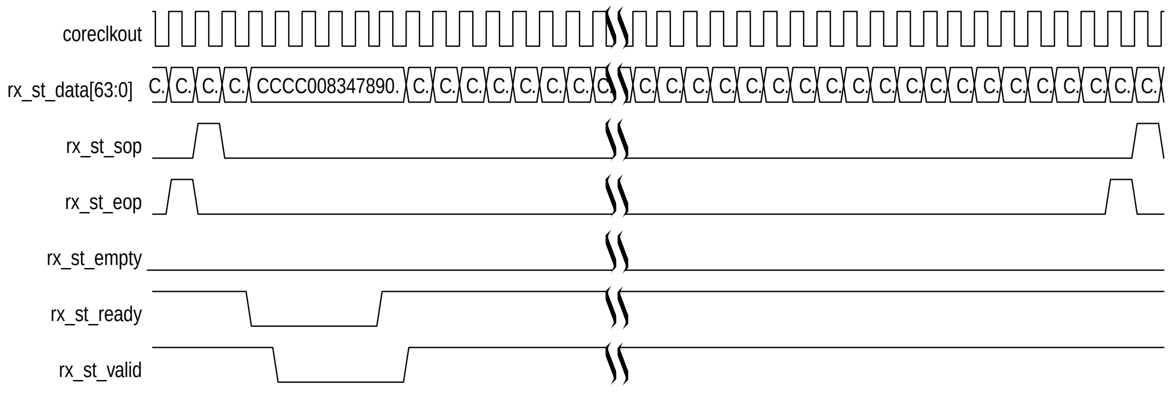


Figure 5–8 illustrates back-to-back transmission on the 64-bit Avalon-ST RX interface with no idle cycles between the assertion of `rx_st_eop` and `rx_st_sop`.

Figure 5–8. 64-Bit Avalon-ST Interface Back-to-Back Transmission



Data Alignment and Timing for the 128-Bit Avalon-ST RX Interface

Figure 5–9 shows the mapping of 128-bit Avalon-ST RX packets to PCI Express TLPs for TLPs with a three dword header and qword aligned addresses.

Figure 5–9. 128-Bit Avalon-ST rx_st_data<*n*> Cycle Definition for 3-Dword Header TLP with Qword Aligned Address

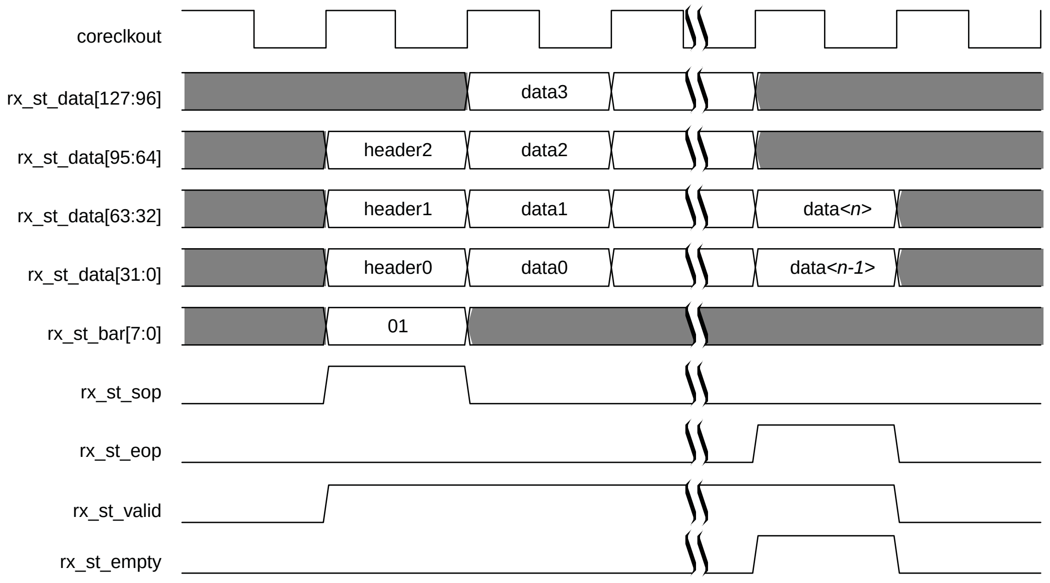


Figure 5–10 shows the mapping of 128-bit Avalon-ST RX packets to PCI Express TLPs for TLPs with a 3 dword header and non-qword aligned addresses. In this case, bits[127:96] represent Data0 because address[2] is set.

Figure 5–10. 128-Bit Avalon-ST rx_st_data<*n*> Cycle Definition for 3-Dword Header TLP with non-Qword Aligned Address

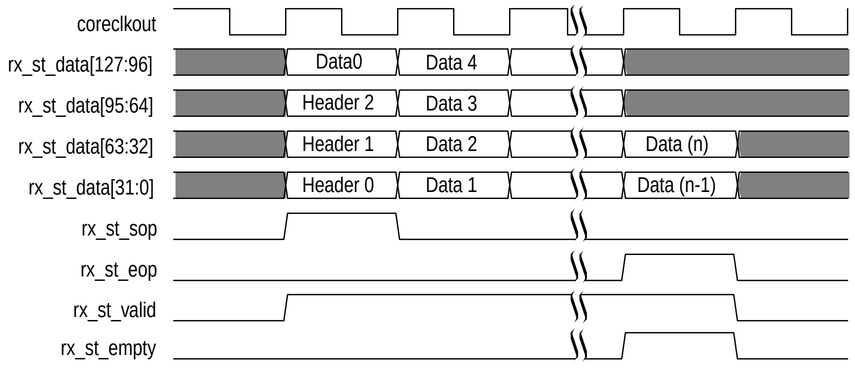


Figure 5–11 shows the mapping of 128-bit Avalon-ST RX packets to PCI Express TLPs for a four dword header with non-qword aligned addresses. In this example, `rx_st_empty` is low because the data ends in the upper 64 bits of `rx_st_data`.

Figure 5–11. 128-Bit Avalon-ST rx_st_data Cycle Definition for 4-Dword Header TLP with non-Qword Aligned Address

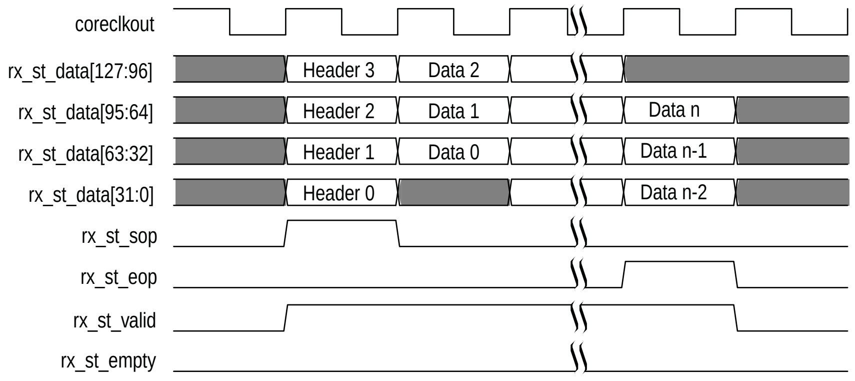


Figure 5–12 shows the mapping of 128-bit Avalon-ST RX packets to PCI Express TLPs for a four dword header with qword aligned addresses.

Figure 5–12. 128-Bit Avalon-ST rx_st_data Cycle Definition for 4-Dword Header TLP with Qword Aligned Address

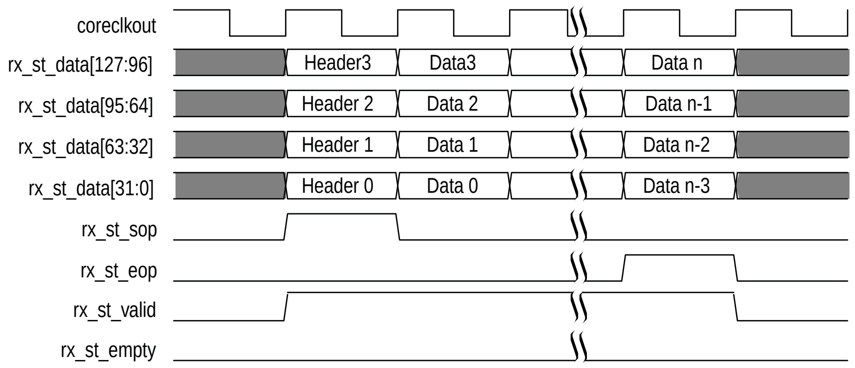


Figure 5–13 illustrates the timing of the RX interface when the Application Layer backpressures the Hard IP by deasserting `rx_st_ready`. The `rx_st_valid` signal must deassert within three cycles after `rx_st_ready` is deasserted. In this example, `rx_st_valid` is deasserted in the next cycle. `rx_st_data` is held until the Application Layer is able to accept it.

Figure 5–13. 128-Bit Application Layer Backpressures Hard IP Transaction Layer for RX Transactions

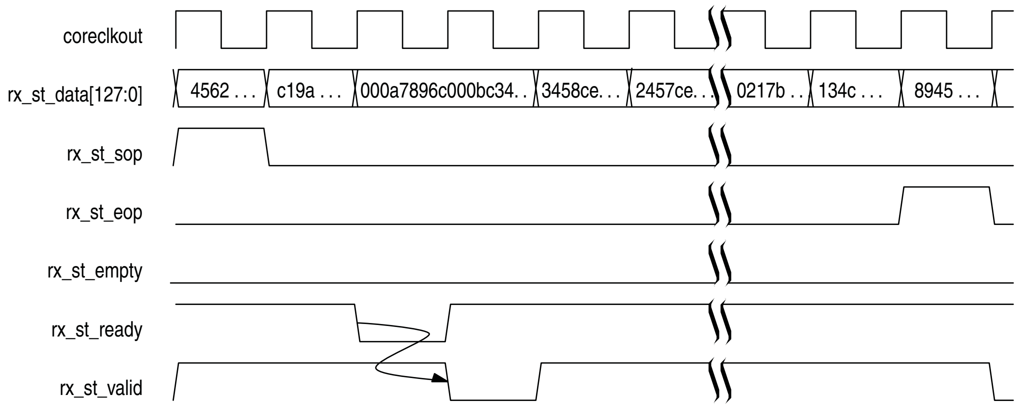


Figure 5–14 illustrates back-to-back transmission on the 128-bit Avalon-ST RX interface with no idle cycles between the assertion of `rx_st_eop` and `rx_st_sop`.

Figure 5–14. 128-Bit Avalon-ST Interface Back-to-Back Transmission

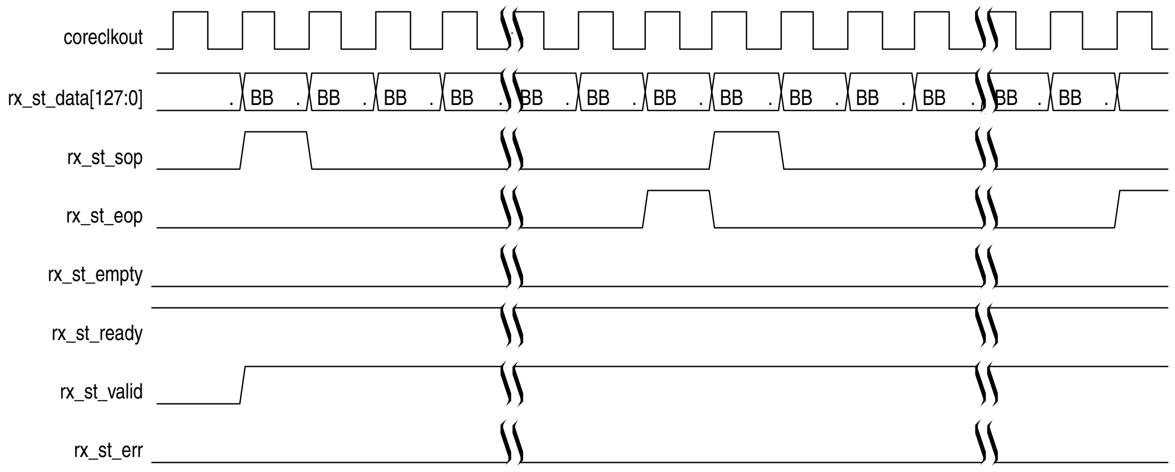
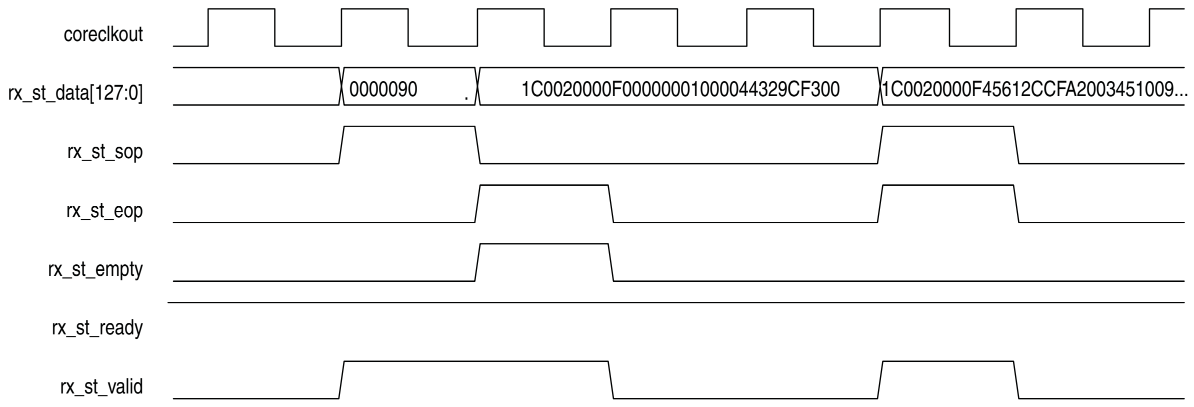


Figure 5–15 illustrates a two-cycle packet with valid data in the lower qword (`rx_st_data[63:0]`) and a one-cycle packet where the `rx_st_sop` and `rx_st_eop` occur in the same cycle.

Figure 5–15. 128-Bit Packet Example Use of rx_st_empty and Single-Cycle Packet



For a complete description of the TLP packet header formats, refer to Appendix A, Transaction Layer Packet (TLP) Header Formats.

Avalon-ST TX Interface

Table 5–4 describes the signals that comprise the Avalon-ST TX Datapath. The TX data signal can be 64 or 128 bits.

Table 5–4. 64- or 128-Bit Avalon-ST TX Datapath (Part 1 of 3)

Signal	Width	Dir	Avalon-ST Type	Description
`tx_st_data`	64, 128	I	`data`	Data for transmission. Transmit data bus. Refer to Figure 5–16 through Figure 5–20 for the mapping of TLP packets to `tx_st_data` and examples of the timing of the 64-bit interface. Refer to Figure 5–21 through Figure 5–26 for the mapping of TLP packets to `tx_st_data` and examples of the timing of the 128-bit interface. The Application Layer must provide a properly formatted TLP on the TX interface. The mapping of message TLPs is the same as the mapping of Transaction Layer TLPs with 4 dword headers. The number of data cycles must be correct for the length and address fields in the header. Issuing a packet with an incorrect number of data cycles results in the TX interface hanging and unable to accept further requests.
`tx_st_sop`	1	I	`start of packet`	Indicates first cycle of a TLP when asserted in the same cycle with `tx_st_valid`.
`tx_st_eop`	1	I	`end of packet`	Indicates last cycle of a TLP when asserted in the same cycle with `tx_st_valid`.
`tx_st_ready` (1)	1	O	`ready`	Indicates that the Transaction Layer is ready to accept data for transmission. The core deasserts this signal to throttle the data stream. `tx_st_ready` may be asserted during reset. The Application Layer should wait at least 2 clock cycles after the reset is released before issuing packets on the Avalon-ST TX interface. The `reset_status` signal can also be used to monitor when the Hard IP has come out of reset. If `tx_st_ready` is asserted by the Transaction Layer on cycle <*n*>, then <*n* + `readyLatency`> is a ready cycle, during which the Application Layer may assert `valid` and transfer data. When `tx_st_ready`, `tx_st_valid` and `tx_st_data` are registered (the typical case), Altera recommends a `readyLatency` of 2 cycles to facilitate timing closure; however, a `readyLatency` of 1 cycle is possible.

Table 5–4. 64- or 128-Bit Avalon-ST TX Datapath (Part 2 of 3)

Signal	Width	Dir	Avalon-ST Type	Description
`tx_st_valid` *(1)*	1	I	`valid`	Clocks `tx_st_data` to the Hard IP when `tx_st_ready` is also asserted. Between `tx_st_sop` and `tx_st_eop`, `tx_st_valid` can be asserted only if `tx_st_ready` is asserted. When `tx_st_ready` deasserts, this signal must deassert within 1 or 2 clock cycles. When `tx_st_ready` reasserts, and `tx_st_data` is in mid-TLP, this signal must reassert within 2 cycles. Refer to Figure 5–19 on page 5–16 for the timing of this signal. To facilitate timing closure, Altera recommends that you register both the `tx_st_ready` and `tx_st_valid` signals. If no other delays are added to the ready-valid latency, the resulting delay corresponds to a `readyLatency` of 2.
`tx_st_empty`	1	I	`empty`	Indicates the number of dwords that are empty during cycles that contain the end of a packet. When asserted, the empty dwords are in the high-order bits. Valid only when `tx_st_eop` is asserted. Not used when `rx_st_data` is 64 bits. When asserted, indicates that the upper qword contains is empty, *does not contain valid data*.
`tx_st_err`	1	I	`error`	Indicates an error on transmitted TLP. This signal is used to nullify a packet. It should only be applied to posted and completion TLPs with payload. To nullify a packet, assert this signal for 1 cycle after the SOP and before the EOP. When a packet is nullified, the following packet should not be transmitted until the next clock cycle. `tx_st_err` is not available for packets that are 1 or 2 cycles long. The error signal must be asserted while the valid signal is asserted.
Component Specific Signals				
`tx_cred_datafccp`	12	O	`component specific`	Data credit limit for the received FC completions. Each credit is 16 bytes.
`tx_cred_datafcnp`	12	O	`component specific`	Data credit limit for the non-posted requests. Each credit is 16 bytes.
`tx_cred_datafcp`	12	O	`component specific`	Data credit limit for the FC posted writes. Each credit is 16 bytes.
`tx_cred_fchipcons`	6	O	`component specific`	Asserted for 1 cycle each time the Hard IP consumes a credit. The 6 bits of this vector correspond to the following 6 types of credit types: ■ [5]: posted headers ■ [4]: posted data ■ [3]: non-posted header ■ [2]: non-posted data ■ [1]: completion header ■ [0]: completion data During a single cycle, the Hard IP can consume either a single header credit or both a header and a data credit.

Table 5–4. 64- or 128-Bit Avalon-ST TX Datapath (Part 3 of 3)

Signal	Width	Dir	Avalon-ST Type	Description
`tx_cred_fc_infinite`	6	O	component specific	When asserted, indicates that the corresponding credit type has infinite credits available and does not need to calculate credit limits. The 6 bits of this vector correspond to the following 6 types of credit types: ■ [5]: posted headers ■ [4]: posted data ■ [3]: non-posted header ■ [2]: non-posted data ■ [1]: completion header ■ [0]: completion data
`tx_cred_hdrfccp`	8	O	component specific	Header credit limit for the FC completions. Each credit is 20 bytes.
`tx_cred_hdrfcnp`	8	O	component specific	Header limit for the non-posted requests. Each credit is 20 bytes.
`tx_cred_hdrfcp`	8	O	component specific	Header credit limit for the FC posted writes. Each credit is 20 bytes.
`ko_cpl_spc_header`	8	O	component specific	The Application Layer can use this signal to build circuitry to prevent RX buffer overflow for completion headers. Endpoints must advertise infinite space for completion headers; however, RX buffer space is finite. `ko_cpl_spc_header` is a static signal that indicates the total number of completion headers that can be stored in the RX buffer.
`ko_cpl_spc_data`	12	O	component specific	The Application Layer can use this signal to build circuitry to prevent RX buffer overflow for completion data. Endpoints must advertise infinite space for completion data; however, RX buffer space is finite. `ko_cpl_spc_data` is a static signal that reflects the total number of 16 byte completion data units that can be stored in the completion RX buffer. The total read data from all outstanding MRd requests must be less than this value to prevent RX FIFO overflow.

Note to Table 5–4:

(1) To be Avalon-ST compliant, your application have a `readyLatency` of 1 or 2 cycles.

Avalon-ST Packets to PCI Express TLPs

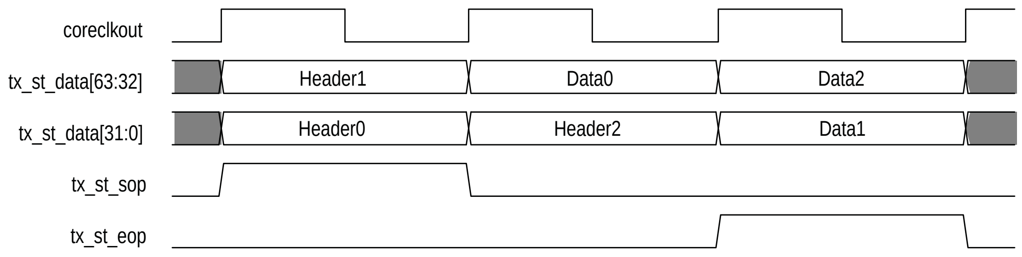
The following figures illustrate the mappings between Avalon-ST packets and PCI Express TLPs. These mappings apply to all types of TLPs, including posted, non-posted, and completion TLPs. Message TLPs use the mappings shown for four dword headers. TLP data is always address-aligned on the Avalon-ST interface whether or not the lower dwords of the header contains a valid address as may be the case with TLP type message request with data payload.

For additional information about TLP packet headers, refer to Appendix A, Transaction Layer Packet (TLP) Header Formats and *Section 2.2.1 Common Packet Header Fields* in the *PCI Express Base Specification 2.1*.

Data Alignment and Timing for the 64-Bit Avalon-ST TX Interface

Figure 5–16 illustrates the mapping between Avalon-ST TX packets and PCI Express TLPs for 3 dword header TLPs with non-qword aligned addresses with a 64-bit bus. (Figure 5–2 on page 5–5 illustrates the storage of non-qword aligned data.) Non-qword aligned addresses occur when address[2] is set. When address[2] is set, `tx_st_data[63:32]`contains `Data0` and `tx_st_data[31:0]` contains dword `header2`.

Figure 5–16. 64-Bit Avalon-ST tx_st_data Cycle Definition for 3-Dword Header TLP with Non-Qword Aligned Address

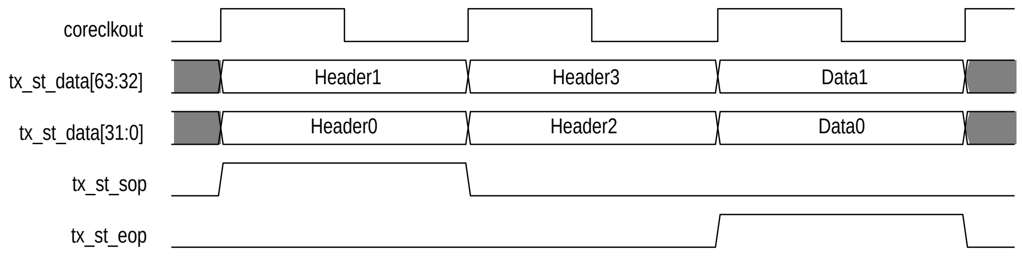


Notes to Figure 5–16:

(1) Header0 ={pcie_hdr_byte0, pcie_hdr _byte1, pcie_hdr _byte2, pcie_hdr _byte3}
(2) Header1 = {pcie_hdr_byte4, pcie_hdr _byte5, header pcie_hdr byte6, pcie_hdr _byte7}
(3) Header2 = {pcie_hdr _byte8, pcie_hdr _byte9, pcie_hdr _byte10, pcie_hdr _byte11}
(4) Data0 = {pcie_data_byte3, pcie_data_byte2, pcie_data_byte1, pcie_data_byte0}
(5) Data1 = {pcie_data_byte7, pcie_data_byte6, pcie_data_byte5, pcie_data_byte4}
(6) Data2 = {pcie_data_byte11, pcie_data_byte10, pcie_data_byte9, pcie_data_byte8}

Figure 5–17 illustrates the mapping between Avalon-ST TX packets and PCI Express TLPs for a four dword header with qword aligned addresses with a 64-bit bus.

Figure 5–17. 64-Bit Avalon-ST tx_st_data Cycle Definition for 4-Dword TLP with Qword Aligned Address



Notes to Figure 5–17:

(1) Header0 = {pcie_hdr_byte0, pcie_hdr _byte1, pcie_hdr _byte2, pcie_hdr _byte3}
(2) Header1 = {pcie_hdr _byte4, pcie_hdr _byte5, pcie_hdr byte6, pcie_hdr _byte7}
(3) Header2 = {pcie_hdr _byte8, pcie_hdr _byte9, pcie_hdr _byte10, pcie_hdr _byte11}
(4) Header3 = pcie_hdr _byte12, pcie_hdr _byte13, header_byte14, pcie_hdr _byte15}, 4 dword header only
(5) Data0 = {pcie_data_byte3, pcie_data_byte2, pcie_data_byte1, pcie_data_byte0}
(6) Data1 = {pcie_data_byte7, pcie_data_byte6, pcie_data_byte5, pcie_data_byte4}

Figure 5–18 illustrates the mapping between Avalon-ST TX packets and PCI Express TLPs for four dword header with non-qword aligned addresses with a 64-bit bus.

Figure 5–18. 64-Bit Avalon-ST tx_st_data Cycle Definition for TLP 4-Dword Header with Non-Qword Aligned Address

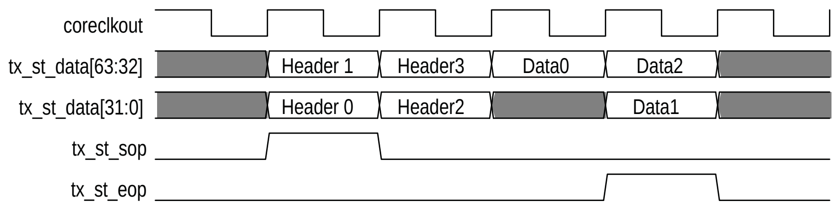


Figure 5–19 illustrates the timing of the TX interface when the Arria V Hard IP for PCI Express IP core backpressures the Application Layer by deasserting `tx_st_ready`. Because the `readyLatency` is two cycles, the Application Layer deasserts `tx_st_valid` after two cycles and holds `tx_st_data` until two cycles after `tx_st_ready` is asserted.

Figure 5–19. 64-Bit Transaction Layer Backpressures the Application Layer

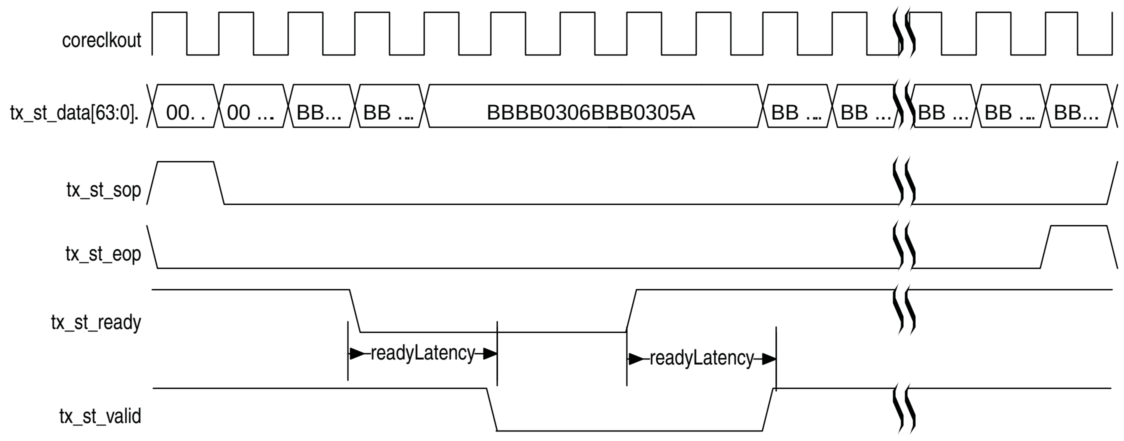
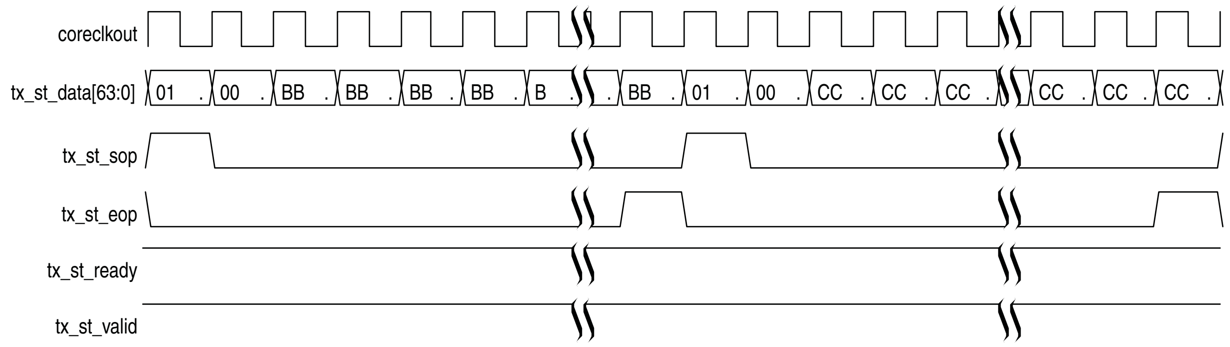


Figure 5–20 illustrates back-to-back transmission of 64-bit packets with no intervening dead cycles between the assertion of `tx_st_eop` and `tx_st_sop`.

Figure 5–20. 64-Bit Back-to-Back Transmission on the TX Interface



Data Alignment and Timing for the 128-Bit Avalon-ST TX Interface

Figure 5–21 shows the mapping of 128-bit Avalon-ST TX packets to PCI Express TLPs for a three dword header with qword aligned addresses.

Figure 5–21. 128-Bit Avalon-ST tx_st_data Cycle Definition for 3-Dword Header TLP with Qword Aligned Address

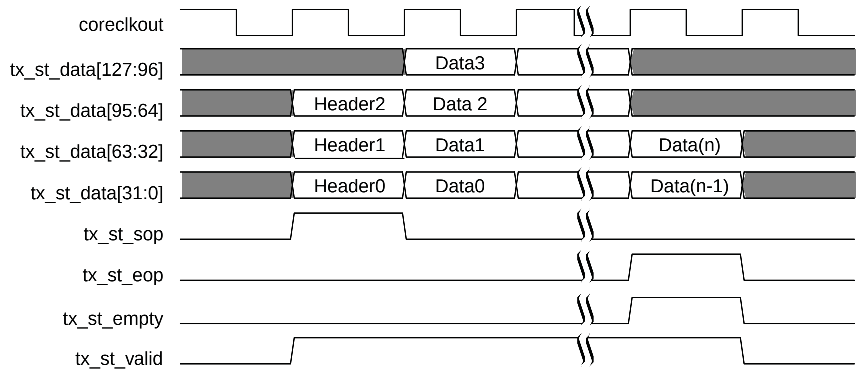


Figure 5–22 shows the mapping of 128-bit Avalon-ST TX packets to PCI Express TLPs for a 3 dword header with non-qword aligned addresses. It also shows `tx_st_err` assertion.

Figure 5–22. 128-Bit Avalon-ST tx_st_data Cycle Definition for 3-Dword Header TLP with non-Qword Aligned Address

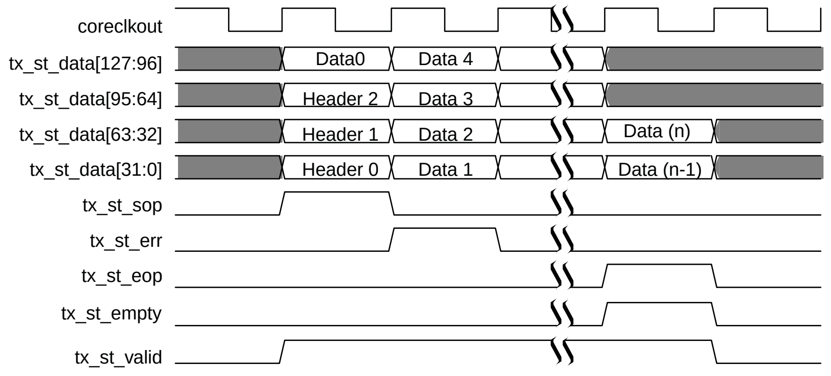


Figure 5–23 shows the mapping of 128-bit Avalon-ST TX packets to PCI Express TLPs for a four dword header TLP with qword aligned data.

Figure 5–23. 128-Bit Avalon-ST tx_st_data Cycle Definition for 4-Dword Header TLP with Qword Aligned Address

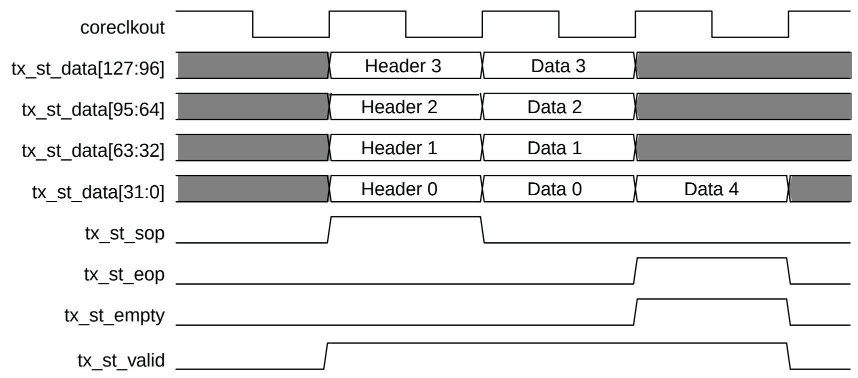


Figure 5–24 shows the mapping of 128-bit Avalon-ST TX packet s to PCI Express TLPs for a four dword header TLP with non-qword aligned addresses. In this example, `tx_st_empty` is low because the data ends in the upper 64 bits of `tx_st_data`.

Figure 5–24. 128-Bit Avalon-ST tx_st_data Cycle Definition for 4-Dword Header TLP with non-Qword Aligned Address

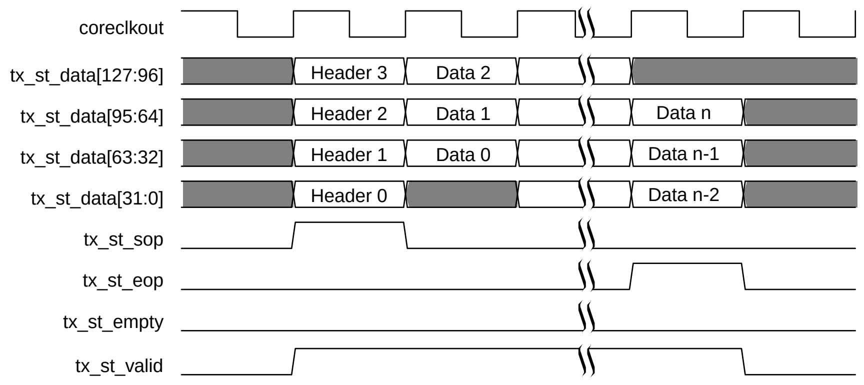


Figure 5–25 illustrates back-to-back transmission of 128-bit packets with no idle cycles between the assertion of `tx_st_eop` and `tx_st_sop`.

Figure 5–25. 128-Bit Back-to-Back Transmission on the Avalon-ST TX Interface

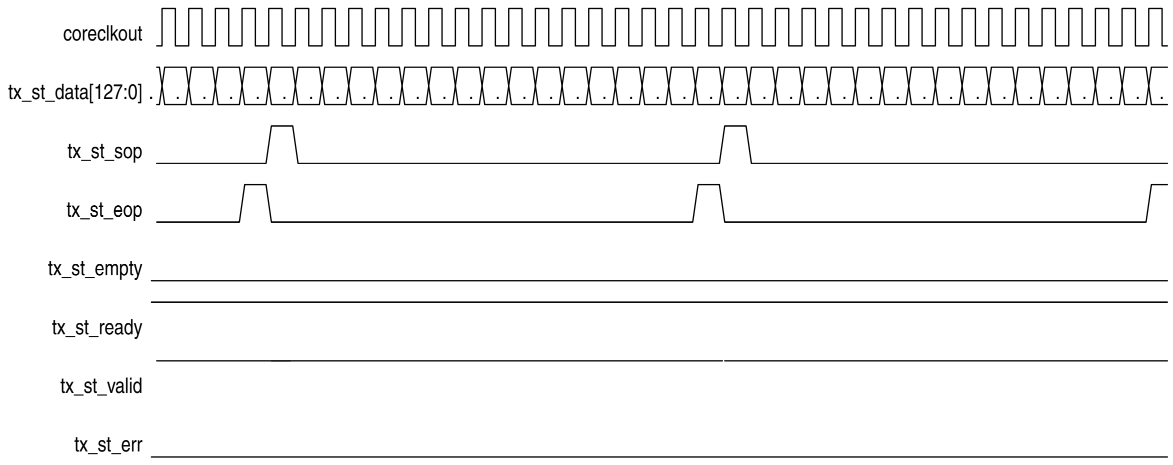
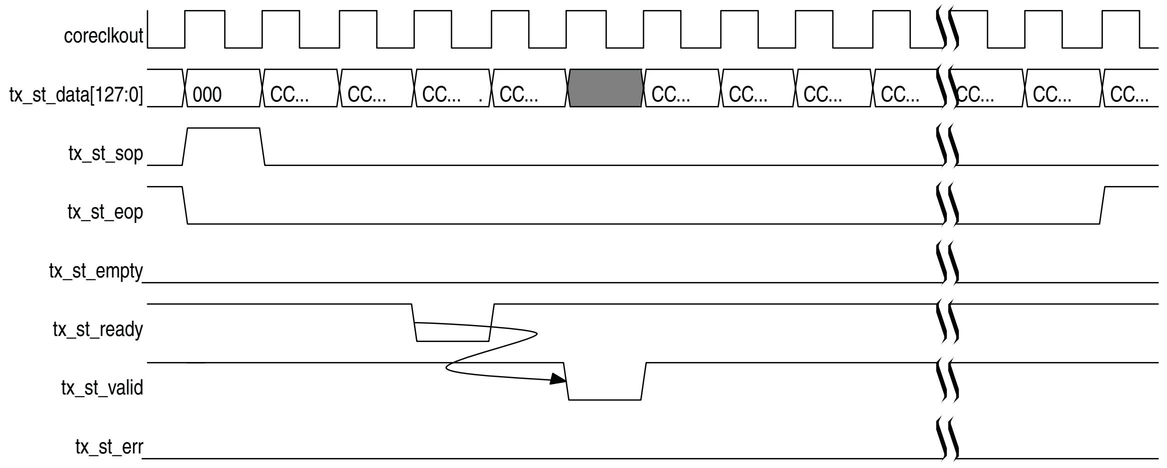


Figure 5–26 illustrates the timing of the TX interface when the Arria V Hard IP for PCI Express IP core backpressures the Application Layer by deasserting `tx_st_ready`. Because the `readyLatency` is two cycles, the Application Layer deasserts `tx_st_valid` after two cycles and holds `tx_st_data` until two cycles after `tx_st_ready` is reasserted

Figure 5–26. 128-Bit Hard IP Backpressures the Application Layer



Root Port Mode Configuration Requests

If your Application Layer implements ECRC forwarding, it should not apply ECRC forwarding to Configuration Type 0 packets that it issues on the Avalon-ST interface. There should be no ECRC appended to the TLP, and the `TD` bit in the TLP header should be set to 0. These packets are processed internally by the Hard IP block and are not transmitted on the PCI Express link.

ECRC Forwarding

On the Avalon-ST interface, the ECRC field follows the same alignment rules as payload data. For packets with payload, the ECRC is appended to the data as an extra dword of payload. For packets without payload, the ECRC field follows the address alignment as if it were a one dword payload. Depending on the address alignment, Figure 5–5 on page 5–6 through Figure 5–12 on page 5–10illustrate the position of the ECRC data for RX data. Figure 5–16 on page 5–15 through Figure 5–24 on page 5–18 illustrate the position of ECRC data for TX data. For packets with no payload data, the ECRC corresponds to the position of Data0 in these figures.

Clock Signals

Table 5–5 describes the clock signals that comprise the clock interface.

Table 5–5. Clock Signals Hard IP Implementation (1)

Signal	I/O	Description
`refclk`	I	Reference clock for the Arria V Hard IP for PCI Express. It must have the frequency specified under the **System Settings** heading in the parameter editor.
`pld_clk`	I	Clocks the Application Layer. You must drive this clock with `coreclkout`.
`coreclkout`	O	This is a fixed frequency clock used by the Data Link and Transaction Layers. To meet PCI Express link bandwidth constraints, this clock has minimum frequency requirements as listed in Table 7–1 on page 7–3.

Note to Table 5–5:

(1) Figure 7–2 on page 7–2 illustrates these clock signals.

Refer to Chapter 7, Reset and Clocks for more information about the clock interface.

Reset Signals

Table 5–6 describes the reset signals.

Table 5–6. Reset and Link Training Signals (Part 1 of 3)

Signal	I/O	Description
`npor`	I	Active high reset signal. `npor` is an input to the embedded reset controller in Arria V devices. You can control this reset input with software. This signal is available when you select the Gen2 lane rate.
`reset_status`	O	Reset Status signal. When asserted, this signal indicates that the Hard IP clock is in reset. The `reset_status` signal is synchronous to the `pld_clk` clock and is deasserted only when the `pld_clk` clock is stable. You should use `reset_status` to drive the reset of your application.

Table 5–6. Reset and Link Training Signals (Part 2 of 3)

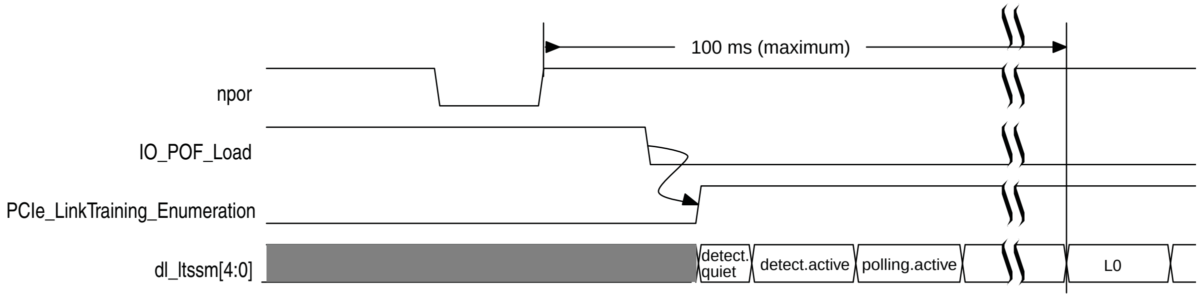
Signal	I/O	Description
`pin_perstn`	I	Active low reset from the PCIe reset pin of the device. This reset signal is an input to the embedded reset controller for PCI Express in Arria V devices. It resets the datapath and control registers. This signal is required for CvP. Arria V devices have 1 or 2 instances of the Hard IP for PCI Express. Each instance has its own `pin_perst` signal. Every Arria V devices has 2 `nPERST` pins, even devices with fewer than 2 instances of the Hard IP for PCI Express. These pins have the following locations: ■ `nPERSTL0`: bottom left Hard IP and CvP blocks ■ `nPERSTL1`: top left Hard IP block For maximum use of the Arria V device, Altera recommends that you use the bottom left Hard IP first. This is the only location that supports CvP over a PCIe link. Refer to the appropriate Arria V device pinout for correct pin assignment for more detailed information about these pins. The *PCI Express Card Electromechanical Specification 2.0* specifies this pin to require 3.3 V. You can drive this 3.3V signal to the `pin_perst` pin even if the V_{CCIO} of the bank is not 3.3V if the following 2 conditions are met: ■ The input signal meets the V_{IH} and V_{IL} specification for LVTTL. ■ The input signal meets the overshoot specification for 100°C operation as specified by the “Maximum Allowed Overshoot and Undershoot Voltage” section in the *Device Datasheet for Arria V Devices* in volume 1 of the *Arria Device Handbook*. Refer to Figure 5–27 on page 5–23 for a timing diagram illustrating the use of this signal.
`serdes_pll_locked`	O	When asserted, indicates that the PLL that generates the `coreclkout_hip` clock signal is locked. In pipe simulation mode this signal is always asserted.
`fixedclk_locked`	O	When asserted, indicates that the PLL that provides the fixed clock required for transceiver initialization is locked. The Application Layer should be held in reset until `fixedclk_locked` is asserted.
`pld_core_ready`	I	When asserted, indicates that the Application Layer is ready for operation and is providing a stable clock to the `pld_clk` input. If the `coreclkout_hip` clock is the direct input to the `pld_clk` input, this input can be connected to the `serdes_pll_locked` output. If a different clock drives `pld_clk`, the corresponding lock signal for that clock should be used to drive this input.
`pld_clk_inuse`	O	When asserted, indicates that the Hard IP Transaction Layer is using the `pld_clk` as its clock and is ready for operation with the Application Layer. For reliable operation, hold the Application Layer in reset until `pld_clk_inuse` is asserted.
`dlup`	O	When asserted, indicates that the Hard IP block is in the DLCSM DLUP state.
`dlup_exit`	O	This signal is active for one `pld_clk` cycle when the IP core exits the DLCSM DLUP state. This signal should cause the Application Layer to assert a global reset. This signal is active low and otherwise remains high.
`ev128ns`	O	Asserted every 128 ns to create a time base aligned activity.
`ev1us`	O	Asserted every 1 µs to create a time base aligned activity.
`hotrst_exit`	O	Hot reset exit. This signal is asserted for 1 clock cycle when the LTSSM exits the hot reset state. This signal should cause the Application Layer to assert a global reset. This signal is active low and otherwise remains high.
`l2_exit`	O	L2 exit. This signal is active low and otherwise remains high. It is asserted for one cycle (changing value from 1 to 0 and back to 1) after the LTSSM transitions from l2_idl to detect.

Table 5–6. Reset and Link Training Signals (Part 3 of 3)

Signal	I/O	Description
`currentspeed[1:0]`	O	Indicates the current speed of the PCIe link. The following encodings are defined: ■ 2b'00: Reserved ■ 2b'01: Gen1 ■ 2b'10: Gen2 ■ 2b'11: Gen3
`dl_ltssm[4:0]`	O	LTSSM state: The LTSSM state machine encoding defines the following states: ■ 00000: detect.quiet ■ 00001: detect.active ■ 00010: polling.active ■ 00011: polling.compliance ■ 00100: polling.configuration ■ 00101: polling.speed ■ 00110: config.linkwidthstart ■ 00111: config.linkaccept ■ 01000: config.lanenumaccept ■ 01001: config.lanenumwait ■ 01010: config.complete ■ 01011: config.idle ■ 01100: recovery.rcvlock ■ 01101: recovery.rcvconfig ■ 01110: recovery.idle ■ 01111: L0 ■ 10000: disable ■ 10001: loopback.entry ■ 10010: loopback.active ■ 10011: loopback.exit ■ 10100: hot.reset ■ 10101: L0s ■ 10110: L1.entry ■ 10111: L1.idle ■ 11000: L2.idle ■ 11001: L2.transmit.wake ■ 11010: recovery.speed

Figure 5–27 illustrates the timing relationship between `npor` and the LTSSM L0s state.

Figure 5–27. 100 ms Requirement



ECC Error Signals

Table 5–7 describes the ECC error signals. Refer to Chapter 11, Interrupts for detailed information about all interrupt mechanisms.

Table 5–7. ECC Error Signals for Hard IP Implementation (1)

Signal	I/O	Description
`derr_cor_ext_rcv0`	O	Indicates a corrected error in the RX buffer. This signal is for debug only. It is not valid until the RX buffer is filled with data.
`derr_cor_ext_rcv1`	O	This signal is not used and can be tied to ground.
`derr_rpl`	O	Indicates an uncorrectable error in the retry buffer. This signal is for debug only.
`derr_cor_ext_rpl`	O	Indicates a corrected ECC error in the retry buffer. This signal is for debug only.

Note to Table 5–7:

(1) The Avalon-ST `rx_st_err` described in Table 5–2 on page 5–3 indicates an uncorrectable error in the RX buffer.

Interrupts for Endpoints

Table 5–8 describes the IP core's interrupt signals for Endpoints. These signals are level sensitive. Refer to Chapter 10, Interrupts for descriptions of all interrupt mechanisms.

Table 5–8. Interrupt Signals for Endpoints (Part 1 of 2)

Signal	I/O	Description
`tl_app_msi_req`	I	Application Layer MSI request. Assertion causes an MSI posted write TLP to be generated based on the MSI configuration register values and the `tl_app_msi_tc` and `app_msi_num` input ports.
`tl_app_msi_ack`	O	Application Layer MSI acknowledge. This signal acknowledges the Application Layer's request for an MSI interrupt.
`tl_app_msi_tc[2:0]`	I	Application Layer MSI traffic class. This signal indicates the traffic class used to send the MSI (unlike INTX interrupts, any traffic class can be used to send MSIs).

Table 5–8. Interrupt Signals for Endpoints (Part 2 of 2)

Signal	I/O	Description
`tl_app_msi_num[4:0]`	I	MSI number of the Application Layer. This signal provides the low order message data bits to be sent in the message data field of MSI messages requested by tl_app_msi_req. Only bits that are enabled by the MSI Message Control register apply. Refer to Table 5–15 on page 5–33 for more information.
`tl_app_msi_func [2:0]`	I	Indicates which function is asserting an interrupt with 0 corresponding to function 0, 1 corresponding to function 1, and so on.
`tl_app_int<a>_sts`	I	Controls legacy interrupts. Assertion of tl_app_int<a>_sts causes an Assert_INTA message TLP to be generated and sent upstream. Deassertion of tl_app_int<a>_sts causes a Deassert_INTA message TLP to be generated and sent upstream.
`tl_app_int<a>_ack`	O	This signal is the acknowledge for tl_app_int<a>_sts. This signal is asserted for at least one cycle either when the `Assert_INTA` message TLP has been transmitted in response to the assertion of the tl_app_int<a>_sts signal or when the `Deassert_INTA` message TLP has been transmitted in response to the deassertion of the tl_app_int<a>_sts signal. Refer to Figure 10–5 on page 10–4 and Figure 10–6 on page 10–4 for timing information.
`tl_app_int<a>_func_ num[2:0]`	O	Indicates which function is asserting an interrupt with 0 corresponding to function 0, 1 corresponding to function 1, and so on.

Interrupts for Root Ports

Table 5–9 describes the signals available to a Root Port to handle interrupts.

Table 5–9. Interrupt Signals for Root Ports

Signal	I/O	Description
`int_status[3:0]`	O	These signals drive legacy interrupts to the Application Layer as follows: ■ int_status[0]: interrupt signal A ■ int_status[1]: interrupt signal B ■ int_status[2]: interrupt signal C ■ int_status[3]: interrupt signal D
`aer_msi_num[4:0]`	I	Advanced error reporting (AER) MSI number. This signal is used by AER to determine the offset between the base message data and the MSI to send.
`pex_msi_num[4:0]`	I	Power management MSI number. This signal is used by power management and hot plug or both to determine the offset between the base message interrupt number and the message interrupt number to send through MSI.
`serr_out`	O	System Error: This signal only applies to Root Port designs that report each system error detected, assuming the proper enabling bits are asserted in the `Root Control` register and the `Device Control` register. If enabled, `serr_out` is asserted for a single clock cycle when a system error occurs. System errors are described in the *PCI Express Base Specification 1.1* or *2.0.* in the `Root Control register`.

Completion Side Band Signals

Table 5–10 describes the signals that comprise the completion side band signals for the Avalon-ST interface. The Arria V Hard IP for PCI Express provides a completion error interface that the Application Layer can use to report errors, such as programming model errors. When the Application Layer detects an error, it can assert the appropriate `cpl_err` bit to indicate what kind of error to log. The Hard IP sets the appropriate status bits for the errors in the Configuration Space, and automatically sends error messages in accordance with the *PCI Express Base Specification*. Note that the Application Layer is responsible for sending the completion with the appropriate completion status value for non-posted requests. Refer to Chapter 12, Error Handling for information on errors that are automatically detected and handled by the Hard IP.

For a description of the completion rules, the completion header format, and completion status field values, refer to Section 2.2.9 of the *PCI Express Base Specification, Rev. 2.1*.

Table 5–10. Completion Signals for the Avalon-ST Interface (Part 1 of 2)

Signal	I/O	Description
`cpl_err[6:0]`	I	Completion error. This signal reports completion errors to the Configuration Space. When an error occurs, the appropriate signal is asserted for one cycle. ■ `cpl_err[0]`: Completion timeout error with recovery. This signal should be asserted when a master-like interface has performed a non-posted request that never receives a corresponding completion transaction after the 50 ms timeout period when the error is correctable. The Hard IP automatically generates an advisory error message that is sent to the Root Complex. ■ `cpl_err[1]`: Completion timeout error without recovery. This signal should be asserted when a master-like interface has performed a non-posted request that never receives a corresponding completion transaction after the 50 ms time-out period when the error is not correctable. The Hard IP automatically generates a non-advisory error message that is sent to the Root Complex. ■ `cpl_err[2]`:Completer abort error. The Application Layer asserts this signal to respond to a posted or non-posted request with a Completer Abort (CA) completion. In the case of a non-posted request, the Application Layer generates and sends a completion packet with Completer Abort (CA) status to the requestor and then asserts this error signal to the Hard IP. The Hard IP automatically sets the error status bits in the Configuration Space register and sends error messages in accordance with the *PCI Express Base Specification, Rev. 2.1.* ■ `cpl_err[3]`: Unexpected completion error. This signal must be asserted when an Application Layer master block detects an unexpected completion transaction. Many cases of unexpected completions are detected and reported internally by the Transaction Layer. For a list of these cases, refer to "Transaction Layer Errors" on page 12–3. ■ `cpl_err[4]`: Unsupported Request (UR) error for posted TLP. The Application Layer asserts this signal to treat a posted request as an Unsupported Request. The Hard IP automatically sets the error status bits in the Configuration Space register and sends error messages in accordance with the *PCI Express Base Specification*. Many cases of Unsupported Requests are detected and reported internally by the Transaction Layer. For a list of these cases, refer to "Transaction Layer Errors" on page 12–3.

Table 5–10. Completion Signals for the Avalon-ST Interface (Part 2 of 2)

Signal	I/O	Description
`cpl_err[6:0]` (continued)		■ `cpl_err[5]`: Unsupported Request error for non-posted TLP. The Application Layer asserts this signal to respond to a non-posted request with an Unsupported Request (UR) completion. In this case, the Application Layer sends a completion packet with the Unsupported Request status back to the requestor, and asserts this error signal. The Hard IP automatically sets the error status bits in the Configuration Space Register and sends error messages in accordance with the *PCI Express Base Specification*. Many cases of Unsupported Requests are detected and reported internally by the Transaction Layer. For a list of these cases, refer to "Transaction Layer Errors" on page 12–3. ■ `cpl_err[6]`: Log header. If header logging is required, this bit must be set in every cycle in which any of `cpl_err[2]`, `cpl_err[3]`, `cpl_err[4]`, or `cpl_err[5]`is asserted. The Application Layer presents the header to the Hard IP by writing the following values to the following 4 registers using LMI before asserting `cpl_err[6]`: ■ lmi_addr: 12'h81C, `lmi_din`: `err_desc_func0[127:96]` ■ lmi_addr: 12'h820, `lmi_din`: `err_desc_func0[95:64]` ■ lmi_addr: 12'h824, `lmi_din`: `err_desc_func0[63:32]` ■ lmi_addr: 12'h828, `lmi_din`: `err_desc_func0[31:0]` Refer to the "LMI Signals" on page 5–33 for more information about LMI signalling.
`cpl_pending`	I	Completion pending. The Application Layer must assert this signal when a master block is waiting for completion, for example, when a transaction is pending.
`cpl_err_func[2:0]`	I	Specifies the function number of the completion error.

Transaction Layer Configuration Space Signals

Table 5–11 describes the Transaction Layer Configuration Space signals.

Table 5–11. Configuration Space Signals (Hard IP Implementation) (Part 1 of 2)

Signal	Width	Dir	Description
`tl_cfg_add`	7	0	Address of the register that has been updated. This signal is an index indicating which Configuration Space register information is being driven onto `tl_cfg_ctl`. The indexing is defined inTable 5–13 on page 5–30.The index increments on every `pld_clk` cycle. The index consists of the following 2 pars: ■ [6:4] - the function number ■ [3:0] - the register address
`tl_cfg_ctl`	32	0	The `tl_cfg_ctl` signal is multiplexed and contains the contents of the Configuration Space registers. The information presented on this bus depends on the `tl_cfg_add` index according toTable 5–13 on page 5–30.
`tl_cfg_ctl_wr`	1	0	Write signal. This signal toggles when `tl_cfg_ctl` has been updated (every 8 `core_clk` cycles). The toggle edge marks where the `tl_cfg_ctl` data changes. You can use this edge as a reference to determine when the data is safe to sample.

Table 5–11. Configuration Space Signals (Hard IP Implementation) (Part 2 of 2)

Signal	Width	Dir	Description
`tl_cfg_sts`	123	0	Configuration status bits. This information updates every `pld_clk` cycle. Bits[52:0] record status information for function0. Bits[62:53] record information for function1. Bits[72:63] record information for function 2, and so on. Refer to Table 5–12 for a detailed description of the status bits.
`tl_cfg_sts_wr`	1	0	Write signal. This signal toggles when `tl_cfg_sts` has been updated (every 8 `core_clk` cycles). The toggle marks the edge where `tl_cfg_sts` data changes. You can use this edge as a reference to determine when the data is safe to sample.
`hpg_ctrler`	5	I	The `hpg_ctrler` signals are only available in Root Port mode and when the Slot Capability register is enabled. Refer to the Use Slot register parameter in Table 3–5 on page 3–6. For Endpoint variations the `hpg_ctrler` input should be hardwired to 0s. The bits have the following meanings:
	[0]	I	Attention button pressed. This signal should be asserted when the attention button is pressed. If no attention button exists for the slot, this bit should be hardwired to 0, and the `Attention Button Present` bit (bit[0]) in the Slot Capability registerparameter should be set to 0.
	[1]	I	Presence detect. This signal should be asserted when a presence detect circuit detects a presence change in the slot.
	[2]	I	Manually-operated retention latch (MRL) sensor changed. This signal should be asserted when an MRL sensor indicates that the MRL is Open. If an MRL Sensor does not exist for the slot, this bit should be hardwired to 0, and the `MRL Sensor Present` bit (bit[2]) in the Slot Capability register parameter should be set to 0.
	[3]	I	Power fault detected. This signal should be asserted when the power controller detects a power fault for this slot. If this slot has no power controller, this bit should be hardwired to 0, and the `Power Controller Present` bit (bit[1]) in the Slot Capability register parameter should be set to 0.
	[4]	I	Power controller status. This signal is used to set the command completed bit of the `Slot Status` register. Power controller status is equal to the power controller control signal. If this slot has no power controller, this bit should be hardwired to 0 and the `Power Controller Present` bit (bit[1]) in the Slot Capability register should be disabled.

Table 5–12 describes the bits of the `tl_cfg_sts` bus for all eight functions and the correspondence between the `tl_cfg_sts` bus and the `cfg_devcsr_func<n>` `cfg_prmcsr_func<n>` registers. Refer to Table 5–13 on page 5–30 for the layout of the configuration control and status information.

Table 5–12. Mapping Between tl_cfg_sts and Configuration Space Registers (Part 1 of 2)

tl_cfg_sts[122:0]	Correspondence	Description
func1[62:59] func2[72:69] func3[82:79] func4[92:89] func5[102:99] func6[112:109] func7[122:119]	`cfg_devcsr_func<n>[19:16]`	Records the following errors: ■ Unsupported request ■ Fatal error ■ Non-fatal error ■ Correctable error
func1[58:54] func2[68:64] func3[78:74] func4[88:84] func5[98:94] func6[108:104] func7[118:114]	`cfg_prmcsr_func<n>[31:27]`	Link status bits as follows: ■ Link autonomous bandwidth status ■ Link bandwidth management status ■ Data Link Layer link active ■ Slot clock configuration
func1[53] func2[63] func3[73] func4[83] func5[93] func6[103] func7[113]	`cfg_prmcsr_func<n>[24]`	6th primary command status error bit. Master data parity error.
[52:49]	`cfg_devcsr_func0[19:16]`	Records the following errors: ■ Unsupported request ■ Fatal error ■ Non-fatal error ■ Correctable error
[48]	`cfg_slotcsr[24]`	Data Link Layer state changed.
[47]	`cfg_slotcsr[20]`	Command completed. (The hot plug controller completed a command.)
[46:31]	`cfg_linkcsr_func0[31:16]`	Records the following link status information: ■ Link autonomous bandwidth status ■ Link bandwidth management status ■ Data Link Layer link active ■ Slot clock configuration ■ Link Training ■ Undefined ■ Negotiated Link Width (6 bits) ■ Link Speed (4 bits)
[30]	`cfg_link2csr_func0[16]`	Records the current de-emphasis level.

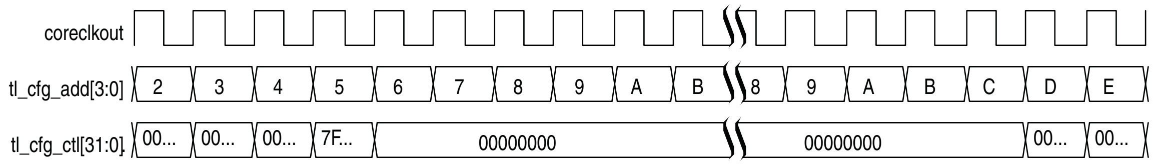
Table 5–12. Mapping Between tl_cfg_sts and Configuration Space Registers (Part 2 of 2)

tl_cfg_sts[122:0]	Correspondence	Description
[29:25]	`cfg_prmcsr_func0[31:27]`	Records the following 5 primary command status errors: ■ Detected parity error ■ Received system error ■ Received master abort ■ Received target abort ■ Signalled target abort.
[24]	`cfg_prmcsr_func0[24]`	Primary command status error bit, data parity reported.
[23:6]	`cfg_rootcsr[25:8]`	Records the following PME status information: ■ PME pending ■ PME status ■ PME request ID[15:0]
[5:1]	`cfg_seccsr[31:27]`	Records the following 5 secondary command status errors: ■ Detected parity error ■ Received system error ■ Received master abort ■ Received target abort ■ Signalled target abort
[0]	`cfg_seccsr[24]`	6th primary command status error bit.

Configuration Space Register Access Timing

Figure 5–28 shows typical traffic on the `tl_cfg_ctl` bus. The `tl_cfg_add` index increments on the rising edge of `coreclkout`, specifying which Configuration Space register information is being driven onto `tl_cfg_ctl`.

Figure 5–28. tl_cfg_ctl Timing



Configuration Space Register Access

The `tl_cfg_ctl` signal is a multiplexed bus that contains the contents of Configuration Space registers as shown in Table 5–11. Information stored in the Configuration Space is accessed in round robin order where `tl_cfg_add` indicates which register is being accessed. Table 5–13 shows the layout of configuration information that is multiplexed on `tl_cfg_ctl`.

Table 5–13. Multiplexed Configuration Register Information Available on tl_cfg_ctl (1)

Address	31:24	23:16	15:8	7:0
0	`cfg_devcsr_func<n>[15:0]` `cfg_devcsr[14:12]=` `Max Read Req Size` (2)	`cfg_devcsr[7:5]=` `Max Payload` (2)	`cfg_dev2csr[15:0]`	
1			`cfg_slotcsr[15:0]`	
2	`cfg_linkscr[15:0]`		`cfg_link2csr[15:0]`	
3	`8'h00`	`cfg_prmcsr_func<n>[15:0]`		`cfg_rootcsr[7:0]`
4	`cfg_seccsr[15:0]`		`cfg_secbus[7:0]`	`cfg_subbus[7:0]`
5	`cfg_msi_addr[11:0]`	`cfg_io_bas[19:0]`		
6	`cfg_msi_addr[43:32]`	`cfg_io_lim[19:0]`		
7	`8h'00`	`cfg_np_bas[11:0]`	`cfg_np_lim[11:0]`	
8	`cfg_pr_bas[31:0]`			
9	`cfgi_msi_addr[31:12]`			`cfg_pr_bas[43:32]`
A	`cfg_pr_lim[31:0]`			
B	`cfg_msi_addr[63:44]`			`cfg_pr_lim[43:32]`
C	`cfg_pmcsr[31:0]`			
D	`cfg_msixcsr[15:0]`		`cfg_msicsr[15:0]`	
E	`tx_ecrcgen[25],` (3) `rx_ecrccheck[24]`	`cfg_tcvcmap[23:0]`		
F	`cfg_msi_data[15:0]`		`3'b000`	`cfg_busdev[12:0]`

Notes to Table 5–13:

(1) Items in blue are only available for Root Ports.

(2) This field is encoded as specified in Section 7.8.4 of the *PCI Express Base Specification*. (3'b000–3'b101 correspond to 128–4096 bytes).

(3) `rx_ecrccheck` and `tx_ecrcgen` are bit s 24 and 25 of `tl_cfg_ctl`, respectively. (Other bit specifications in this table indicate the bit location within the Configuration Space register.)

Table 5–14 describes the Configuration Space registers referred to in Table 5–11 and Table 5–13.

Table 5–14. Configuration Space Register Descriptions (Part 1 of 3)

Register	Width	Dir	Description	Register Reference
`cfg_devcsr_func<n>`	32	O	`cfg_devcsr_func<n>[31:16]`is status and `cfg_devcsr[15:0]` is Device Control for the PCI Express capability structure.	Table 6–7 on page 6–4
`cfg_dev2csr`	32	O	`cft_dev2csr[31:16]` is status 2 and `cfg_dev2csr[15:0]` is device control 2 for the PCI Express capability structure.	Table 6–8 on page 6–4

Table 5–14. Configuration Space Register Descriptions (Part 2 of 3)

Register	Width	Dir	Description	Register Reference
`cfg_slotcsr`	16	O	`cfg_slotcsr[31:16]` is the Slot Control and `cfg_slotcsr[15:0]`is the Slot Status of the PCI Express capability structure. This register is only available in Root Port mode.	Table 6–7 on page 6–4 Table 6–8 on page 6–4
`cfg_linkcsr`,	32	O	`cfg_linkcsr[15:0]`is the primary Link Control of the PCI Express capability structure.	Table 6–7 on page 6–4 Table 6–8 on page 6–4
`cfg_link2csr`			`cfg_link2csr[31:16]` is the secondary Link Status and `cfg_link2csr[15:0]`is the secondary Link Control of the PCI Express capability structure for Gen2 operation. When `tl_cfg_addr=2`, `tl_cfg_ctl` returns the primary and secondary Link Control registers, `{cfg_linkcsr[15:0], cfg_lin2csr[15:0]}`, the primary Link Status register, `cfg_linkcsr[31:16]`, is available on `tl_cfg_sts[46:31]`. For Gen1 variants, the link bandwidth notification bit is always set to 0. For Gen2 variants, this bit is set to 1.	Table 6–8 on page 6–4
`cfg_prmcsr_func<`*n*`>`	16	O	Base/Primary Control and Status register for the PCI Configuration Space.	Table 6–2 on page 6–2 0x004 (Type 0) Table 6–3 on page 6–2 0x004 (Type 1)
`cfg_rootcsr`	8	O	Root Control and Status register of the PCI-Express capability. This register is only available in Root Port mode.	Table 6–7 on page 6–4 Table 6–8 on page 6–4
`cfg_seccsr`	16	O	Secondary bus Control and Status register of the PCI-Express capability. This register is only available in Root Port mode.	Table 6–3 on page 6–2 0x01C
`cfg_secbus`	8	O	Secondary bus number. Available in Root Port mode.	Table 6–3 on page 6–2 0x018
`cfg_subbus`	8	O	Subordinate bus number. Available in Root Port mode.	Table 6–3 on page 6–2 0x018
`cfg_io_bas`	20	O	The upper 20 bits of the IO limit registers of the Type1 Configuration Space. This register is only available in Root Port mode.	Table 6–3 on page 6–2 0x01C
`cfg_io_lim`	20	O	The upper 20 bits of the IO limit registers of the Type1 Configuration Space. This register is only available in Root Port mode.	Table 6–8 on page 6–4 0x01C
`cfg_np_bas`	12	O	The upper 12 bits of the memory base register of the Type1 Configuration Space. This register is only available in Root Port mode.	Table 3–7 on page 3–8 EXP ROM
`cfg_np_lim`	12	O	The upper 12 bits of the memory limit register of the Type1 Configuration Space. This register is only available in Root Port mode.	Table 3–7 on page 3–8 EXP ROM

Table 5–14. Configuration Space Register Descriptions (Part 3 of 3)

Register	Width	Dir	Description	Register Reference
`cfg_pr_bas`	44	O	The upper 44 bits of the prefetchable base registers of the Type1 Configuration Space. This register is only available in Root Port mode.	Table 6–3 on page 6–2 0x024 and Table 3–7 on page 3–8 Prefetchable memory
`cfg_pr_lim`	44	O	The upper 44 bits of the prefetchable limit registers of the Type1 Configuration Space. Available in Root Port mode.	Table 6–3 on page 6–2 0x024 and Table 3–7 on page 3–8 Prefetchable memory
`cfg_pmcsr`	32	O	`cfg_pmcsr[31:16]` is Power Management Control and `cfg_pmcsr[15:0]`is the Power Management Status register.	Table 6–6 on page 6–4 0x07C
`cfg_msixcsr`	16	O	MSI-X message control.	Table 6–5 on page 6–3 0x068
`cfg_msicsr`	16	O	MSI message control. Refer to Table 5–15 for the fields of this register.	Table 6–4 on page 6–3 0x050
`cfg_tcvcmap`	24	O	Configuration traffic class (TC)/virtual channel (VC) mapping. The Application Layer uses this signal to generate a TLP mapped to the appropriate channel based on the traffic class of the packet. `cfg_tcvcmap[2:0]`: Mapping for TC0 (always 0). `cfg_tcvcmap[5:3]`: Mapping for TC1. `cfg_tcvcmap[8:6]`: Mapping for TC2. `cfg_tcvcmap[11:9]`: Mapping for TC3. `cfg_tcvcmap[14:12]`: Mapping for TC4. `cfg_tcvcmap[17:15]`: Mapping for TC5. `cfg_tcvcmap[20:18]`: Mapping for TC6. `cfg_tcvcmap[23:21]`: Mapping for TC7.	—
`cfg_busdev`	13	O	Bus/Device Number captured by or programmed in the Hard IP.	Table A–5 on page A–2 0x08

Table 5–15 shows the layout of the Configuration MSI Control Status register.

Table 5–15 describes the use of the various fields of the Configuration MSI Control and Status Register.

Table 5–15. Configuration MSI Control Status Register Field Descriptions

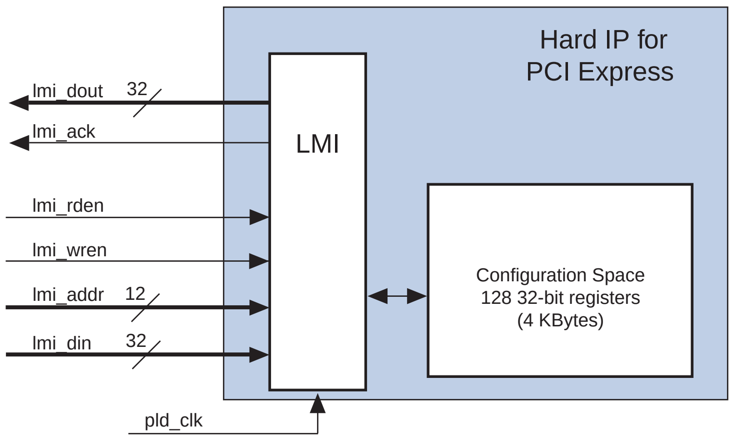
Bit(s)	Field	Description
[15:9]	`reserved`	—
[8]	`mask capability`	Per vector masking capable. This bit is hardwired to 0 because the functions do not support the optional MSI per vector masking using the `Mask_Bits` and `Pending_Bits` registers defined in the *PCI Local Bus Specification, Rev. 3.0*. Per vector masking can be implemented using Application Layer registers.
[7]	`64-bit address capability`	64-bit address capable ■ 1: function capable of sending a 64-bit message address ■ 0: function not capable of sending a 64-bit message address
[6:4]	`multiples message enable`	Multiple message enable: This field indicates permitted values for MSI signals. For example, if "100" is written to this field 16 MSI signals are allocated ■ 000: 1 MSI allocated ■ 001: 2 MSI allocated ■ 010: 4 MSI allocated ■ 011: 8 MSI allocated ■ 100: 16 MSI allocated ■ 101: 32 MSI allocated ■ 110: Reserved ■ 111: Reserved
[3:1]	`multiple message capable`	Multiple message capable: This field is read by system software to determine the number of requested MSI messages. ■ 000: 1 MSI requested ■ 001: 2 MSI requested ■ 010: 4 MSI requested ■ 011: 8 MSI requested ■ 100: 16 MSI requested ■ 101: 32 MSI requested ■ 110: Reserved
[0]	`MSI Enable`	If set to 0, this component is not permitted to use MSI.

LMI Signals

LMI interface is used to write log error descriptor information in the TLP header log registers. The LMI access to other registers is intended for debugging, not normal operation.

Figure 5–29 illustrates the LMI interface.

Figure 5–29. Local Management Interface



The LMI interface is synchronized to `pld_clk` and runs at frequencies up to 250 MHz. The LMI address is the same as the Configuration Space address. The read and write data are always 32 bits. The LMI interface provides the same access to Configuration Space registers as Configuration TLP requests. Register bits have the same attributes, (read only, read/write, and so on) for accesses from the LMI interface and from Configuration TLP requests.

When a LMI write has a timing conflict with configuration TLP access, the configuration TLP accesses have higher priority. LMI writes are held and executed when configuration TLP accesses are no longer pending. An acknowledge signal is sent back to the Application Layer when the execution is complete.

All LMI reads are also held and executed when no configuration TLP requests are pending. The LMI interface supports two operations: local read and local write. The timing for these operations complies with the Avalon-MM protocol described in the *Avalon Interface Specifications*. LMI reads can be issued at any time to obtain the contents of any Configuration Space register. LMI write operations are not recommended for use during normal operation. The Configuration Space registers are written by requests received from the PCI Express link and there may be unintended consequences of conflicting updates from the link and the LMI interface. LMI Write operations are provided for AER header logging, and debugging purposes only.

CAUTION In Root Port mode, do not access the Configuration Space using TLPs and the LMI bus simultaneously.

Table 5–16 describes the signals that comprise the LMI interface.

Table 5–16. LMI Interface (Part 1 of 2)

Signal	Width	Dir	Description
`lmi_dout`	32	O	Data outputs
`lmi_rden`	1	I	Read enable input
`lmi_wren`	1	I	Write enable input
`lmi_ack`	1	O	Write execution done/read data valid

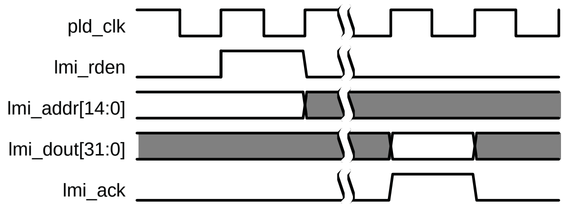
Table 5–16. LMI Interface (Part 2 of 2)

Signal	Width	Dir	Description
`lmi_addr`	15	I	Address inputs, [1:0] not used
`lmi_din`	32	I	Data inputs

LMI Read Operation

Figure 5–30 illustrates the read operation.

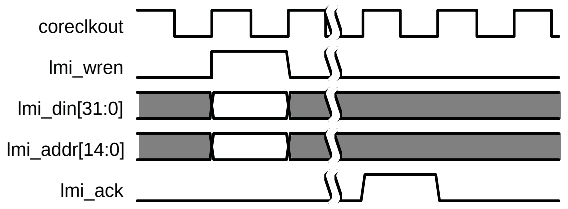
Figure 5–30. LMI Read



LMI Write Operation

Figure 5–31 illustrates the LMI write. Only writeable configuration bits are overwritten by this operation. Read-only bits are not affected. LMI write operations are not recommended for use during normal operation with the exception of AER header logging.

Figure 5–31. LMI Write



Power Management Signals

Table 5–17 describes the power management signals.

Table 5–17. Power Management Signals

Signal	I/O	Description
pme_to_cr	I	Power management turn off control register. Root Port—When this signal is asserted, the Root Port sends the PME_turn_off message. Endpoint—This signal is asserted to acknowledge the PME_turn_off message by sending pme_to_ack to the Root Port.
pme_to_sr	O	Power management turn off status register. Root Port—This signal is asserted for 1 clock cycle when the Root Port receives the pme_turn_off acknowledge message. Endpoint—This signal is asserted for 1 cycle when the Endpoint receives the PME_turn_off message from the Root Port.
pm_event	I	Power Management Event. This signal is only available for Endpoints. The Endpoint initiates a a power_management_event message (PM_PME) that is sent to the Root Port. If the Hard IP is in a low power state, the link exists from the low-power state to send the message. This signal is positive edge-sensitive.
pm_event_func[2:0]	I	Specifies the function associated with a Power Management Event.
pm_data[9:0]	I	Power Management Data. This bus indicates power consumption of the component. This bus can only be implemented if all three bits of AUX_power (part of the Power Management Capabilities structure) are set to 0. This bus includes the following bits: ■ pm_data[9:2]: Data Register: This register maintains a value associated with the power consumed by the component. (Refer to the example below) ■ pm_data[1:0]: Data Scale: This register maintains the scale used to find the power consumed by a particular component and can include the following values: b’00: unknown b’01: 0.1 × b’10: 0.01 × b’11: 0.001 × For example, the two registers might have the following values: ■ pm_data[9:2]: b’1110010 = 114 ■ pm_data[1:0]: b’10, which encodes a factor of 0.01 To find the maximum power consumed by this component, multiply the data value by the data Scale (114 × .01 = 1.14). 1.14 watts is the maximum power allocated to this component in the power state selected by the data_select field.
pm_auxpwr	I	Power Management Auxiliary Power: This signal can be tied to 0 because the L2 power state is not supported.

Table 5–18 shows the layout of the Power Management Capabilities register.

Table 5–18. Power Management Capabilities Register

31 24	22 16	15	14 13	12 9	8	7 2	1 0
`data register`	`rsvd`	`PME_status`	`data_scale`	`data_select`	`PME_EN`	`rsvd`	`PM_state`

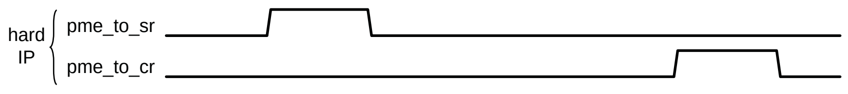
Table 5–19 describes the use of the various fields of the Power Management Capabilities register.

Table 5–19. Power Management Capabilities Register Field Descriptions

Bits	Field	Description
[31:24]	`Data register`	This field indicates in which power states a function can assert the `PME# message`.
[22:16]	`reserved`	—
[15]	`PME_status`	When set to 1, indicates that the function would normally assert the `PME#` message independently of the state of the `PME_en` bit.
[14:13]	`data_scale`	This field indicates the scaling factor when interpreting the value retrieved from the data register. This field is read-only.
[12:9]	`data_select`	This field indicates which data should be reported through the data register and the `data_scale` field.
[8]6	`PME_EN`	1: indicates that the function can assert PME# 0: indicates that the function cannot assert PME#
[7:2]	`reserved`	—
[1:0]	`PM_state`	Specifies the power management state of the operating condition being described. The following encodings are defined: ■ 2b'00 D0 ■ 2b'01 D1 ■ 2b'10 D2 ■ 2b'11 D3 A device returns 2b'11 in this field and `Aux` or `PME Aux` in the `type` register to specify the *D3-Cold PM* state. An encoding of 2b'11 along with any other `type` register value specifies the *D3-Hot* state.

Figure 5–32 illustrates the behavior of `pme_to_sr` and `pme_to_cr` in an Endpoint. First, the Hard IP receives the `PME_turn_off` message which causes `pme_to_sr` to assert. Then, the Application Layer sends the `PME_to_ack` message to the Root Port by asserting `pme_to_cr`.

Figure 5–32. pme_to_sr and pme_to_cr in an Endpoint IP core



Physical Layer Interface Signals

This section describes the global PHY support signals for the internal PHY. The MegaWizard Plug-In Manager generates a SERDES variation file, <*variation*>**_serdes.<v** or **vhd** >, in addition of the Hard IP variation file, <*variation*>.<**v** or **vhd**>. For Arria V GX devices the SERDES entity is included in the library files for PCI Express.

Transceiver Reconfiguration

Table 5–20 describes the transceiver support signals. In Table 5–20, <*n*> is the number of lanes.

Table 5–20. Transceiver Control Signals

Signal Name	I/O	Description

`reconfig_fromxcvr[(<n>70)-1:0]`
`reconfig_toxcvr[(<n>46)-1:0]`
	O	These are the parallel transceiver dynamic reconfiguration buses. Dynamic reconfiguration is required to compensate for variations due to process, voltage and temperature (PVT). Among the analog settings that you can reconfigure are: $\mathbf{V}_{OD}$, pre-emphasis, and equalization.
		You can use the Altera Transceiver Reconfiguration Controller to dynamically reconfigure analog settings in Arria V devices. For more information about instantiating the Altera Transceiver Reconfiguration Controller IP core refer to Chapter 13, Reconfiguration and Offset Cancellation.

`busy_xcvr_reconfig`	O	When asserted, indicates that the a reconfiguration operation is in progress.

For more information about the refer to the “Transceiver Reconfiguration Controller” chapter in the *Altera Transceiver PHY IP Core User Guide*.

The following sections describe signals for the serial or parallel PIPE interfaces. The PIPE interface is only available for simulation.

Serial Interface Signals

Table 5–21 describes the serial interface signals.

Table 5–21. 1-Bit Interface Signals

Signal	I/O	Description
`tx_out[<n-1>:0]` (1)	O	Transmit input. These signals are the serial outputs.
`rx_in[<n-1>:0]` (1)	I	Receive input. These signals are the serial inputs.

Note to Table 5–21:

(1) <*n*> = 1 for the ×1 IP core. <*n*> = 4 for the ×4 IP core. <*n*> = 8 for the ×8 IP core.

Refer to Pin-out Files for Altera Devices for pin-out tables for all Altera devices in **.pdf**, **.txt**, and **.xls** formats.

Figure 5–33 shows the channel placement for ×1 and ×4 variants.

Figure 5–33. Channel Placement for ×1 and ×4 Variants

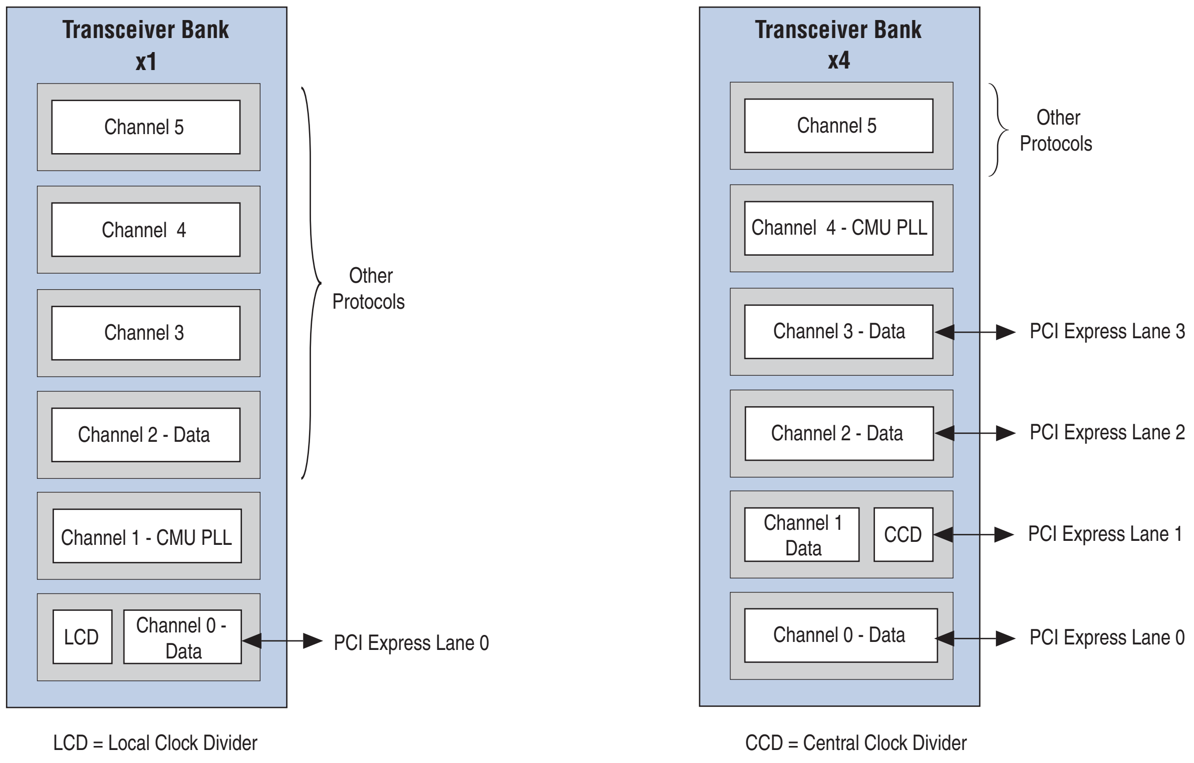
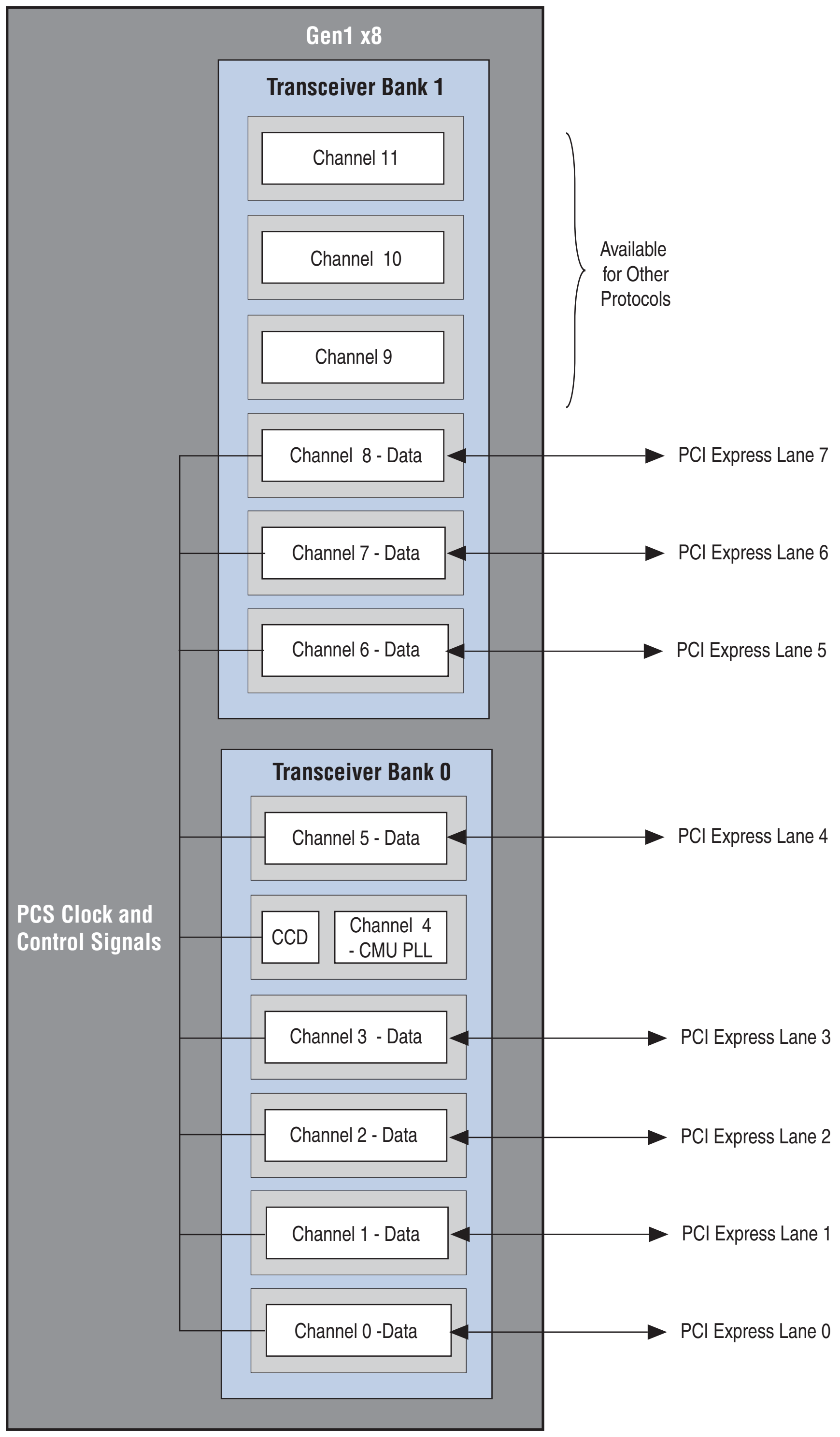


Figure 5–34 shows the channel placement for ×8 variants.

Figure 5–34. Channel Placement for ×8 Variants



CCD = Central Clock Divider

PIPE Interface Signals

The PIPE signals are available so that you can simulate using either the one-bit or the PIPE interface. Simulation is much faster using the PIPE interface. You can use the 8-bit PIPE interface for simulation even though your actual design includes the serial interface to the internal transceivers. However, it is not possible to use the Hard IP PIPE interface in an actual device. Table 5–22 describes the PIPE interface signals used for a standard 16-bit SDR or 8-bit SDR interface. In Table 5–22, signals that include lane number 0 also exist for lanes 1-7. In Qsys, the signals that are part of the PIPE interface have the prefix, *hip_pipe*. The signals which are included to simulate the PIPE interface have the prefix, *hip_pipe_sim_pipe*.

Table 5–22. PIPE Interface Signals (Part 1 of 2)

Signal	I/O	Description
`txdata0[15:0]`	O	Transmit data <*n*>. This bus transmits data on lane <*n*>.
`txdatak0[1:0]` (1)	O	Transmit data control <*n*>. This signal serves as the control bit for `txdata<`*n*`>`.
`txdetectrx0` (1)	O	Transmit detect receive <*n*>. This signal tells the PHY layer to start a receive detection operation or to begin loopback.
`txelecidle` (1)	O	Transmit electrical idle <*n*>. This signal forces the TX output to electrical idle.
`txcompl0` (1)	O	Transmit compliance <*n*>. This signal forces the running disparity to negative in compliance mode (negative COM character).
`rxpolarity0` (1)	O	Receive polarity <*n*>. This signal instructs the PHY layer to invert the polarity of the 8B/10B receiver decoding block.
`powerdown0[1:0]` (1)	O	Power down <*n*>. This signal requests the PHY to change its power state to the specified state (P0, P0s, P1, or P2).
`tx_deemph0`	O	Transmit de-emphasis selection. The Arria V Hard IP for PCI Express sets the value for this signal based on the indication received from the other end of the link during the Training Sequences (TS). You do not need to change this value.
`rxdata0[15:0]` (1) (2)	I	Receive data <*n*>. This bus receives data on lane <*n*>.
`rxdatak0[1:0]` (1) (2)	I	Receive data control <*n*>. This signal separates control and data symbols.
`rxvalid0` (1) (2)	I	Receive valid <*n*>. This symbol indicates symbol lock and valid data on `rxdata<`*n*`>` and `rxdatak<`*n*`>`.
`phystatus0` (1) (2)	I	PHY status <*n*>. This signal communicates completion of several PHY requests.
`eidleinfersel0[2:0]`	O	Electrical idle entry inference mechanism selection. The following encodings are defined: ■ 3'b0xx: Electrical Idle Inference not required in current LTSSM state ■ 3'b100: Absence of COM/SKP Ordered Set the in 128 us window for Gen1 or Gen2 ■ 3'b101: Absence of TS1/TS2 Ordered Set in a 1280 UI interval for Gen1 or Gen2 ■ 3'b110: Absence of Electrical Idle Exit in 2000 UI interval for Gen1 and 16000 UI interval for Gen2 ■ 3'b111: Absence of Electrical idle exit in 128 us window for Gen1

Table 5–22. PIPE Interface Signals (Part 2 of 2)

Signal	I/O	Description
`rxelecidle0` (1) (2)	I	Receive electrical idle <*n*>. This signal forces the receive output to electrical idle.
`rxstatus0[2:0]` (1) (2)	I	Receive status <*n*>. This signal encodes receive status and error codes for the receive data stream and receiver detection.
`ltssmstate0[4:0]`	O	LTSSM state: The LTSSM state machine encoding defines the following states: ■ 00000: detect.quiet ■ 00001: detect.active ■ 00010: polling.active ■ 00011: polling.compliance ■ 00100: polling.configuration ■ 00101: polling.speed ■ 00110: config.linkwidthstart ■ 00111: config.linkaccept ■ 01000: config.lanenumaccept ■ 01001: config.lanenumwait ■ 01010: config.complete ■ 01011: config.idle ■ 01100: recovery.rcvlock ■ 01101: recovery.rcvconfig ■ 01110: recovery.idle ■ 01111: L0 ■ 10000: disable ■ 10001: loopback.entry ■ 10010: loopback.active ■ 10011: loopback.exit ■ 10100: hot.reset ■ 10101: LOs ■ 11001: L2.transmit.wake ■ 11010: speed.recovery
`sim_pipe_rate`	O	Specifies the lane rate. The 2-bit encodings have the following meanings: ■ 2'b00–reserved ■ 2'b01: Gen1 rate (2.5 Gbps) ■ 2'b10: Gen2 rate (5.0 Gbps) ■ 2'b11–Gen3 rate (8.0 Gbps)
`sim_pipe_pclk_in`	I	This clock is used for PIPE simulation only, and is derived from the `refclk`. It is the PIPE interface clock used for PIPE mode simulation.

Notes to Table 5–22:

(1) Signals that include lane number 0 also exist for lanes 1-7.

(2) These signals are for simulation only. For Quartus II software compilation, these pipe signals can be left floating.

Test Signals

The `test_in` bus provides run-time control and monitoring of the internal state of the Arria V Hard IP for PCI Express. Table 5–23 describes the test signals.

CAUTION Altera recommends that you use the `test_in` signals for debug or non-critical status monitoring purposes such as LED displays of PCIe link status. They should not be used for design function purposes. Use of these signals will make it more difficult to close timing on the design. The test signals have not been rigorously verified and will not function as documented in some corner cases.

The debug signals provided on `test_out` are not available in the current release.

Table 5–23 describes the `test_in` bus signals. In Qsys these signals have the prefix, *hip_ctrl_*.

Table 5–23. Test Interface Signals (1), (2)

Signal	I/O	Description
`test_in[31:1]`	I	[0]–Simulation mode. This signal can be set to 1 to accelerate initialization by reducing the value of many initialization counters. [4:1] Reserved. These signals are not supported in the current release. You must drive them to all 0's. [6:5] Compliance test mode. Disable/force compliance mode: ■ bit 0–When set, prevents the LTSSM from entering compliance mode. Toggling this bit controls the entry and exit from the compliance state, enabling the transmission of Gen1 and Gen2 compliance patterns. ■ bit 1–Forces compliance mode. Forces entry to compliance mode when timeout is reached in polling.active state (and not all lanes have detected their exit condition). ■ [31:7] Reserved.
	O	When set to 1, the PIPE interface is in simulation mode.
	I	Compliance mode test switch. When set to 1, the IP core is in compliance mode which is used for Compliance Base Board testing (CBB) testing. When set to 0, the IP core is in operates normally. Connect this signal to a switch to turn on and off compliance mode. Refer to the *PCI Express High Performance Reference Design* for an actual coding example to specify CBB tests.
`lane_act[3:0]`		Lane Active Mode: This signal indicates the number of lanes that configured during link training. The following encodings are defined: ■ 4'b0001: 1 lane ■ 4'b0010: 2 lanes ■ 4'b0100: 4 lanes ■ 4'b1000: 8 lanes

Notes to Table 5–23:

(1) All signals are per lane.

(2) Refer to "PIPE Interface Signals" on page 5–41 for definitions of the PIPE interface signals.

6. Register Descriptions

This section describes registers that you can access the PCI Express Configuration Space. It includes the following sections:

- Configuration Space Register Content
- Correspondence between Configuration Space Registers and the PCIe Spec 2.1

Configuration Space Register Content

Table 6–1 shows the common Configuration Space header. The following tables provide more details.

To facilitate finding additional information about these PCI Express registers, the following tables provide the name of the corresponding section in the *PCI Express Base Specification Revision 2.1.*

Table 6–1. Common Configuration Space Header

Byte Offset	Register Set
0x000:0x03C	PCI Type 0 Configuration Space Header (Refer to Table 6–2 for details.)
0x000:0x03C	PCI Type 1 Configuration Space Header (Refer to Table 6–3 for details.)
0x040:0x04C	Reserved.
0x050:0x05C	MSI Capability Structure, Version 1.0a and 1.1 (Refer to Table 6–4 for details.)
0x060:0x064	Reserved
0x068:0x070	MSI-X Capability Structure, Version 2.1 (Refer to Table 6–5 for details.)
0x070:0x074	Reserved
0x078:0x07C	Power Management Capability Structure (Refer to Table 6–6 for details.)
0x080:0x0BC	PCI Express Advanced Error Reporting Extended Capability Structure (Refer to Table 6–8 for details.)
0x0BD:0x0BF	Reserved
0x0C0:0x0C4	Subsystem ID and Subsystem Vendor ID
0x0C8-0x7FC	Reserved
0x800:0x834	Advanced error reporting (AER) (optional)
0x838:0xFFF	Reserved

For comprehensive information about these registers, refer to Chapter 7 of the *PCI Express Base Specification Revision 2.1.*

Table 6–2 describes the Type 0 Configuration settings.

In the following tables, the names of fields that are defined by parameters in the parameter editor are links to the description of that parameter. These links appear as green text.

Table 6–2. PCI Type 0 Configuration Space Header (Endpoints), Rev2.1 Spec: Type 0 Configuration Space Header

Byte Offset	31:24	23:16	15:8	7:0
0x000	Device ID		Vendor ID	
0x004	Status		Command	
0x008	Class code			Revision ID
0x00C	0x00	Header Type (Port type)	0x00	Cache Line Size
0x010	Func0–Func7 BARs and Expansion ROM			
0x014	Func0–Func7 BARs and Expansion ROM			
0x018	Func0–Func7 BARs and Expansion ROM			
0x01C	Func0–Func7 BARs and Expansion ROM			
0x020	Func0–Func7 BARs and Expansion ROM			
0x024	Func0–Func7 BARs and Expansion ROM			
0x028	Reserved			
0x02C	Subsystem Device ID		Subsystem Vendor ID	
0x030	Expansion ROM base address			
0x034	Reserved			Capabilities Pointer
0x038	Reserved			
0x03C	0x00	0x00	Interrupt Pin	Interrupt Line

Note to Table 6–2:

(1) Refer to Table 6–9 on page 6–5 for a comprehensive list of correspondences between the Configuration Space registers and the *PCI Express Base Specification 2.1*.

Table 6–3 describes the Type 1 Configuration settings.

Table 6–3. PCI Type 1 Configuration Space Header (Root Ports) (Part 1 of 2), Rev2.1 Spec: Type 1 Configuration Space Header

Byte Offset	31:24	23:16	15:8	7:0
0x0000	Device ID		Vendor ID	
0x004	Status		Command	
0x008	Class code			Revision ID
0x00C	BIST	Header Type	Primary Latency Timer	Cache Line Size
0x010	Func0–Func7 BARs and Expansion ROM			
0x014	Func0–Func7 BARs and Expansion ROM			
0x018	Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number
0x01C	Secondary Status		I/O Limit	I/O Base

Table 6–3. PCI Type 1 Configuration Space Header (Root Ports) (Part 2 of 2), Rev2.1 Spec: Type 1 Configuration Space Header

Byte Offset	31:24	23:16	15:8	7:0
0x020	`Memory Limit`		`Memory Base`	
0x024	`Prefetchable Memory Limit`		`Prefetchable Memory Base`	
0x028	`Prefetchable Base Upper 32 Bits`			
0x02C	`Prefetchable Limit Upper 32 Bits`			
0x030	`I/O Limit Upper 16 Bits`		`I/O Base Upper 16 Bits`	
0x034	Reserved			`Capabilities Pointer`
0x038	Expansion ROM Base Address			
0x03C	`Bridge Control`		`Interrupt Pin`	`Interrupt Line`

Note to Table 6–3:

(1) Refer to Table 6–9 on page 6–5 for a comprehensive list of correspondences between the Configuration Space registers and the *PCI Express Base Specification 2.0.*

Table 6–4 describes the MSI Capability structure.

Table 6–4. MSI Capability Structure, Rev2.1 Spec: MSI Capability Structures

Byte Offsets (1)	31:24	23:16	15:8	7:0
0x050	`Message Control` Configuration MSI Control Status Register Field Descriptions		`Next Cap Ptr`	`Capability ID`
0x054	`Message Address`			
0x058	`Message Upper Address`			
0x05C	`Reserved`		`Message Data`	

Note to Table 6–4:

(1) Specifies the byte offset within Arria V Hard IP for PCI Express IP core's address space.

(2) Refer to Table 6–9 on page 6–5 for a comprehensive list of correspondences between the Configuration Space registers and the *PCI Express Base Specification 2.0.*

Table 6–5 describes the MSI-X Capability structure.

Table 6–5. MSI-X Capability Structure, Rev2.1 Spec: MSI-X Capability Structures

Byte Offset	31:24	23:16	15:8	7:3	2:0
0x068	`Message Control`		`Next Cap Ptr`	`Capability ID`	
0x06C	`MSI-X Table Offset`				Pending Bit Array (PBA) Offset

Note to Table 6–5:

(1) Refer to Table 6–9 on page 6–5 for a comprehensive list of correspondences between the Configuration Space registers and the *PCI Express Base Specification 2.0.*

Table 6–6 describes the Power Management Capability structure.

Table 6–6. Power Management Capability Structure, Rev2.1 Spec

Byte Offset	31:24	23:16	15:8	7:0
0x078	Capabilities Register		Next Cap PTR	Cap ID
0x07C	Data	PM Control/Status Bridge Extensions	Power Management Status & Control	

Note to Table 6–6:

(1) Refer to Table 6–9 on page 6–5 for a comprehensive list of correspondences between the Configuration Space registers and the *PCI Express Base Specification 2.0.*

Table 6–7 describes the PCI Express AER Extended Capability structure.

Table 6–7. PCI Express Advanced Error Reporting Extended Capability Structure, Rev2.1 Spec: Advanced Error Reporting Capability

Byte Offset	31:24	23:16	15:8	7:0
0x800	PCI Express Enhanced Capability Header			
0x804	Uncorrectable Error Status Register			
0x808	Uncorrectable Error Mask Register			
0x80C	Uncorrectable Error Severity Register			
0x810	Correctable Error Status Register			
0x814	Correctable Error Mask Register			
0x818	Advanced Error Capabilities and Control Register			
0x81C	Header Log Register			
0x82C	Root Error Command			
0x830	Root Error Status			
0x834	Error Source Identification Register		Correctable Error Source ID Register	

Note to Table 6–7:

(1) Refer to Table 6–9 on page 6–5 for a comprehensive list of correspondences between the Configuration Space registers and the *PCI Express Base Specification 2.0.*

Table 6–8 describes the PCI Express Capability Structure.

Table 6–8. PCIe Capability Structure 2.1, Rev2.1 Spec (Part 1 of 2)

Byte Offset	31:16	15:8	7:0
0x080	PCI Express Capabilities Register	Next Cap Pointer	PCI Express Cap ID
0x084	Device Capabilities		
0x088	Device Status	Device Control 2	
0x08C	Link		
0x090	Link Status	Link Control	
0x094	Slot		
0x098	Slot Status	Slot Control	
0x09C	Root Capabilities	Root Control	
0x0A0	Root Status		
0x0A4	Device Capabilities 2		

Table 6–8. PCIe Capability Structure 2.1, Rev2.1 Spec (Part 2 of 2)

Byte Offset	31:16	15:8	7:0
0x0A8	Device Status 2	Device Control 2	
0x0AC	Link Capabilities 2		
0x0B0	Link Status 2	Link Control 2	
0x0B4	Slot Capabilities 2		
0x0B8	Slot Status 2	Slot Control 2	

Note to Table 6–8:

(1) Registers not applicable to a device are reserved.

(2) Refer to Table 6–9 on page 6–5 for a comprehensive list of correspondences between the Configuration Space registers and the *PCI Express Base Specification 2.0.*

Correspondence between Configuration Space Registers and the PCIe Spec 2.1

Table 6–9 provides a comprehensive correspondence between the Configuration Space registers and their descriptions in the *PCI Express Base Specification 2.0.*

Table 6–9. Correspondence Configuration Space Registers and PCIe Base Specification Rev. 2.0 (Part 1 of 3)

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
Table 6-1. Common Configuration Space Header		
0x000:0x03C	PCI Header Type 0 Configuration Registers	Type 0 Configuration Space Header
0x000:0x03C	PCI Header Type 1 Configuration Registers	Type 1 Configuration Space Header
0x040:0x04C	Reserved	
0x050:0x05C	MSI Capability Structure	MSI and MSI-X Capability Structures
0x068:0x070	MSI Capability Structure	MSI and MSI-X Capability Structures
0x070:0x074	Reserved	
0x078:0x07C	Power Management Capability Structure	PCI Power Management Capability Structure
0x080:0x0B8	PCI Express Capability Structure	PCI Express Capability Structure
0x080:0x0B8	PCI Express Capability Structure	PCI Express Capability Structure
0x0B8:0x0FC	Reserved	
0x094:0x0FF	Root Port	
0x100:0x16C	Virtual Channel Capability Structure (Reserved)	Virtual Channel Capability
0x170:0x17C	Reserved	
0x180:0x1FC	Virtual channel arbitration table (Reserved)	VC Arbitration Table
0x200:0x23C	Port VC0 arbitration table (Reserved)	Port Arbitration Table
0x240:0x27C	Port VC1 arbitration table (Reserved)	Port Arbitration Table
0x280:0x2BC	Port VC2 arbitration table (Reserved)	Port Arbitration Table
0x2C0:0x2FC	Port VC3 arbitration table (Reserved)	Port Arbitration Table
0x300:0x33C	Port VC4 arbitration table (Reserved)	Port Arbitration Table
0x340:0x37C	Port VC5 arbitration table (Reserved)	Port Arbitration Table
0x380:0x3BC	Port VC6 arbitration table (Reserved)	Port Arbitration Table

Table 6–9. Correspondence Configuration Space Registers and PCIe Base Specification Rev. 2.0 (Part 2 of 3)

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x3C0:0x3FC	Port VC7 arbitration table (Reserved)	Port Arbitration Table
0x400:0x7FC	Reserved	PCIe spec corresponding section name
0x800:0x834	Advanced Error Reporting AER (optional)	Advanced Error Reporting Capability
0x838:0xFFF	Reserved	
Table 6-2. PCI Type 0 Configuration Space Header (Endpoints), Rev2.1 Spec: Type 0 Configuration Space Header		
0x000	Device ID Vendor ID	Type 0 Configuration Space Header
0x004	Status Command	Type 0 Configuration Space Header
0x008	Class Code Revision ID	Type 0 Configuration Space Header
0x00C	0x00 Header Type 0x00 Cache Line Size	Type 0 Configuration Space Header
0x010	Base Address 0	Base Address Registers (Offset 10h - 24h)
0x014	Base Address 1	Base Address Registers (Offset 10h - 24h)
0x018	Base Address 2	Base Address Registers (Offset 10h - 24h)
0x01C	Base Address 3	Base Address Registers (Offset 10h - 24h)
0x020	Base Address 4	Base Address Registers (Offset 10h - 24h)
0x024	Base Address 5	Base Address Registers (Offset 10h - 24h)
0x028	Reserved	Type 0 Configuration Space Header
0x02C	Subsystem Device ID Subsystem Vendor ID	Type 0 Configuration Space Header
0x030	Expansion ROM base address	Type 0 Configuration Space Header
0x034	Reserved Capabilities PTR	Type 0 Configuration Space Header
0x038	Reserved	Type 0 Configuration Space Header
0x03C	0x00 0x00 Interrupt Pin Interrupt Line	Type 0 Configuration Space Header
Table 6-3. PCI Type 1 Configuration Space Header (Root Ports) , Rev2.1 Spec: Type 1 Configuration Space Header		
0x000	Device ID Vendor ID	Type 1 Configuration Space Header
0x004	Status Command	Type 1 Configuration Space Header
0x008	Class Code Revision ID	Type 1 Configuration Space Header
0x00C	BIST Header Type Primary Latency Timer Cache Line Size	Type 1 Configuration Space Header
0x010	Base Address 0	Base Address Registers (Offset 10h/14h)
0x014	Base Address 1	Base Address Registers (Offset 10h/14h)
0x018	Secondary Latency Timer Subordinate Bus Number Secondary Bus Number Primary Bus Number	Secondary Latency Timer (Offset 1Bh)/Type 1 Configuration Space Header/ /Primary Bus Number (Offset 18h)
0x01C	Secondary Status I/O Limit I/O Base	Secondary Status Register (Offset 1Eh) / Type 1 Configuration Space Header
0x020	Memory Limit Memory Base	Type 1 Configuration Space Header
0x024	Prefetchable Memory Limit Prefetchable Memory Base	Prefetchable Memory Base/Limit (Offset 24h)
0x028	Prefetchable Base Upper 32 Bits	Type 1 Configuration Space Header
0x02C	Prefetchable Limit Upper 32 Bits	Type 1 Configuration Space Header
0x030	I/O Limit Upper 16 Bits I/O Base Upper 16 Bits	Type 1 Configuration Space Header
0x034	Reserved Capabilities PTR	Type 1 Configuration Space Header

Table 6–9. Correspondence Configuration Space Registers and PCIe Base Specification Rev. 2.0 (Part 3 of 3)

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x038	Expansion ROM Base Address	Type 1 Configuration Space Header
0x03C	Bridge Control Interrupt Pin Interrupt Line	Bridge Control Register (Offset 3Eh)
Table 6-4.MSI Capability Structure, Rev2.1 Spec: MSI Capability Structures		
0x050	Message Control Next Cap Ptr Capability ID	MSI and MSI-X Capability Structures
0x054	Message Address	MSI and MSI-X Capability Structures
0x058	Message Upper Address	MSI and MSI-X Capability Structures
0x05C	Reserved Message Data	MSI and MSI-X Capability Structures
Table 6-5. MSI-X Capability Structure, Rev2.1 Spec: MSI-X Capability Structures		
0x68	Message Control Next Cap Ptr Capability ID	MSI and MSI-X Capability Structures
0x6C	MSI-X Table Offset BIR	MSI and MSI-X Capability Structures
0x70	Pending Bit Array (PBA) Offset BIR	MSI and MSI-X Capability Structures
Table 6-6. Power Management Capability Structure, Rev2.1 Spec		
0x078	Capabilities Register Next Cap PTR Cap ID	PCI Power Management Capability Structure
0x07C	Data PM Control/Status Bridge Extensions Power Management Status & Control	PCI Power Management Capability Structure
Table 6-7 PCI Express Advanced Error Reporting Extended Capability Structure, Rev2.1 Spec: Advanced Error Reporting Capability		
0x800	PCI Express Enhanced Capability Header	Advanced Error Reporting Enhanced Capability Header
0x804	Uncorrectable Error Status Register	Uncorrectable Error Status Register
0x808	Uncorrectable Error Mask Register	Uncorrectable Error Mask Register
0x80C	Uncorrectable Error Severity Register	Uncorrectable Error Severity Register
0x810	Correctable Error Status Register	Correctable Error Status Register
0x814	Correctable Error Mask Register	Correctable Error Mask Register
0x818	Advanced Error Capabilities and Control Register	Advanced Error Capabilities and Control Register
0x81C	Header Log Register	Header Log Register
0x82C	Root Error Command	Root Error Command Register
0x830	Root Error Status	Root Error Status Register
0x834	Error Source Identification Register Correctable Error Source ID Register	Error Source Identification Register

7. Reset and Clocks

This chapter covers the functional aspects of the reset and clock circuitry for the Arria V Hard IP for PCI Express. It includes the following sections:

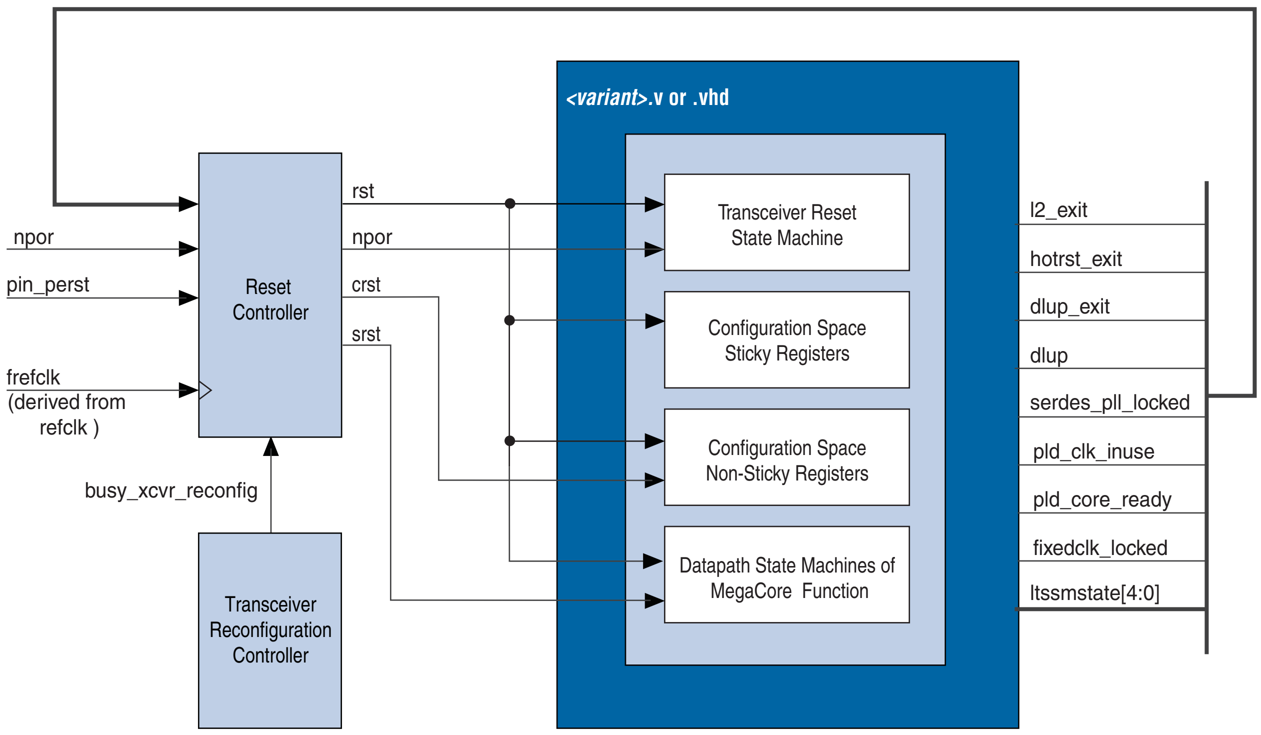
- Reset
- Clocks

For descriptions of the available reset and clock *signals* refer to the following sections: "Reset Signals" on page 5–20, and "Clock Signals" on page 5–20.

Reset

The Arria V Hard IP for PCI Express IP core includes an embedded reset controller to handle the initial reset of the PMA, PCS, and Hard IP for PCI Express IP core. The `pin_perst` signal which is driven from one of the two designated `nPERST` pins of the device initiates reset. Figure 7–1 provides a high-level view of the reset hardware.

Figure 7–1. Reset Controller for Arria V Devices



As Figure 7–1 indicates, the reset controller receives information about the current LTSSM state from the Hard IP and drives global and specialized resets to the Hard IP.

After reset is complete, all port registers and state machines are set to their initialization values with the exception of sticky registers as defined Sections 7.4 and 7.6 of the *PCI Express Base Specification*. The reset outputs of the reset controller affect varying amounts of the state of the Hard IP block as follows:

- `npor` and `pin_perst`—These signals reset all sticky registers that may not be reset in L2 low power mode or by the fundamental reset.
- `srst`—The `srst` signal initiates a synchronous reset of the datapath state machines.
- `crst`—The `crst` signal initiates a synchronous reset of the nonsticky Configuration Space registers.

☞ The Arria V embedded reset sequence meets the 100 ms configuration time specified in the *PCI Express Base Specification 2.1.*

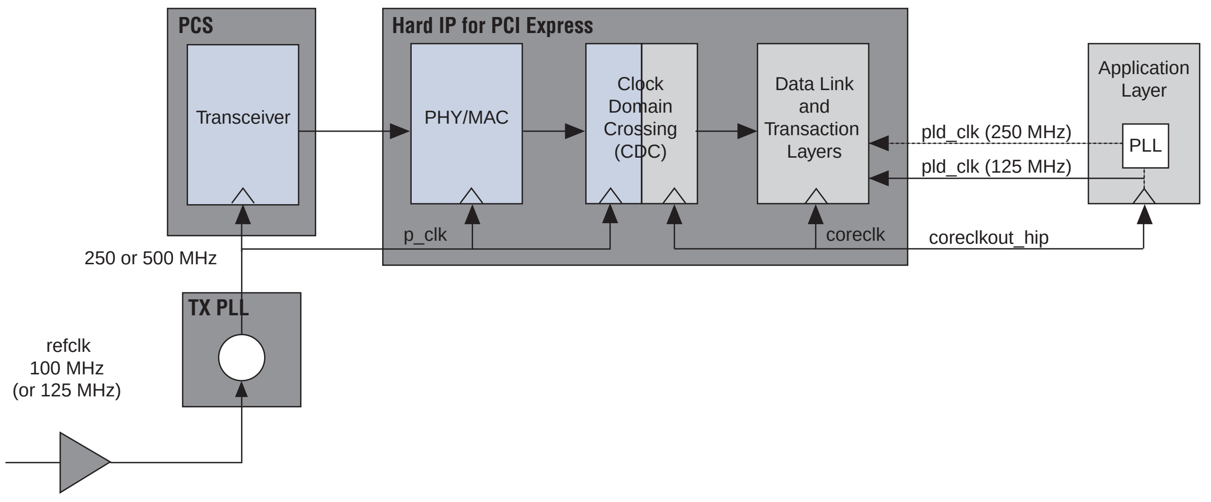
Clocks

In accordance with the *PCI Express Base Specification 2.1*, you must provide a 100 MHz reference clock that is connected directly to the transceiver. As a convenience, you may also use a 125 MHz input reference clock as input to the TX PLL. The output of the transceiver drives `coreclkout_hip`. `coreclkout_hip` must be connected back to the `pld_clk` input clock, possibly through a clock distribution circuit required by the specific application. For Application Layers running at 250 MHz, Altera recommends using a PLL to ease timing closure.

The Hard IP contains a clock domain crossing (CDC) synchronizer at the interface between the PHY/MAC and the DLL layers which allows the Data Link and Transaction Layers to run at frequencies independent of the PHY/MAC and provides more flexibility for the user clock interface. Depending on system requirements, you can use this additional flexibility to enhance performance by running at a higher frequency for latency optimization or at a lower frequency to save power.

Figure 7–2 illustrates the clock domains.

Figure 7–2. Arria V Hard IP for PCI Express Clock Domains



As Figure 7–2 indicates, there are three clock domains:

- p_clk
- coreclkout
- pld_clk

p_clk

The transceiver derives `p_clk` from the 100 MHz `refclk` signal that you must provide to the device. The *PCI Express Base Specification 2.1* requires that the `refclk` signal frequency be 100 MHz ±300 PPM; however, as a convenience, you can also use a reference clock that is 125 MHz ±300 PPM.

The transitions between Gen1 and Gen2 should be glitchless. `p_clk` can be turned off for the entire 1 ms timeout assigned for the PHY to change the clock rate; however, `p_clk` should be stable before the 1 ms timeout expires.

The CDC module implements the asynchronous clock domain crossing between the PHY/MAC `p_clk` domain and the Data Link Layer `coreclk` domain.

coreclkout

The `coreclkout` signal is derived from `p_clk`. Table 7–1 lists frequencies for `coreclkout` which are a function of the link width, data rate, and the width of the Avalon-ST bus.

Table 7–1. coreclkout Values for All Parameterizations

Link Width	Max Link Rate	Avalon Interface Width	coreclkout_hip
×1	Gen1	64	125 MHz
×1	Gen1	64	62.5 MHz (1)
×4	Gen1	64	125 MHz
×8	Gen1	128	125 MHz
×1	Gen2	64	125 MHz
×4	Gen2	128	125 MHz

Note to Table 7–1:

(1) This mode saves power.

The frequencies and widths specified in Table 7–1 are maintained throughout operation. If the link downtrains to a lesser link width or changes to a different maximum link rate, it maintains the frequencies it was originally configured for as specified in Table 7–1. (The Hard IP throttles the interface to achieve a lower throughput.) If the link also downtrains from Gen2 to Gen1, it maintains the frequencies from the original link width, for either Gen1 or Gen2.

pld_clk

This clock drives the Transaction Layer, Data Link Layer, part of the PHY/MAC Layer, and the Application Layer. Ideally, the `pld_clk` drives all user logic in the Application Layer, including other instances of the Arria V Hard IP for PCI Express and memory interfaces. The `pld_clk` input clock pin is typically generated from the `coreclkout_hip` output clock pin.

8. Transaction Layer Protocol (TLP) Details

This chapter provides detailed information about the Arria V Hard IP for PCI Express. TLP handling. It includes the following sections:

- Supported Message Types
- Transaction Layer Routing Rules
- Receive Buffer Reordering

Supported Message Types

Table 8–1 describes the message types supported by the Hard IP.

Table 8–1. Supported Message Types (2) (Part 1 of 3)

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
INTX Mechanism Messages						For Endpoints, only INTA messages are generated.
Assert_INTA	Receive	Transmit	No	Yes	No	For Root Port, legacy interrupts are translated into message interrupt TLPs which triggers the `int_status[3:0]` signals to the Application Layer. ■ `int_status[0]`: Interrupt signal A ■ `int_status[1]`: Interrupt signal B ■ `int_status[2]`: Interrupt signal C ■ `int_status[3]`: Interrupt signal D
Assert_INTB	Receive	Transmit	No	No	No	
Assert_INTC	Receive	Transmit	No	No	No	
Assert_INTD	Receive	Transmit	No	No	No	
Deassert_INTA	Receive	Transmit	No	Yes	No	
Deassert_INTB	Receive	Transmit	No	No	No	
Deassert_INTC	Receive	Transmit	No	No	No	
Deassert_INTD	Receive	Transmit	No	No	No	
Power Management Messages						
PM_Active_State_Nak	Transmit	Receive	No	Yes	No	
PM_PME	Receive	Transmit	No	No	Yes	
PME_Turn_Off	Transmit	Receive	No	No	Yes	The `pme_to_cr` signal sends and acknowledges this message: ■ Root Port: When `pme_to_cr` is asserted, the Root Port sends the PME_turn_off message. ■ Endpoint: When `pme_to_cr` is asserted, the Endpoint acknowledges the `PME_turn_off` message by sending a `pme_to_ack` message to the Root Port.
PME_TO_Ack	Receive	Transmit	No	No	Yes	

Table 8–1. Supported Message Types [2] (Part 2 of 3)

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Error Signaling Messages						
ERR_COR	Receive	Transmit	No	Yes	No	In addition to detecting errors, a Root Port also gathers and manages errors sent by downstream components through the ERR_COR, ERR_NONFATAL, AND ERR_FATAL Error Messages. In Root Port mode, there are two mechanisms to report an error event to the Application Layer: ■ `serr_out` output signal. When set, indicates to the Application Layer that an error has been logged in the AER capability structure ■ `aer_msi_num` input signal. When the **Implement advanced error reporting** option is turned on, you can set `aer_msi_num` to indicate which MSI is being sent to the root complex when an error is logged in the AER Capability structure.
ERR_NONFATAL	Receive	Transmit	No	Yes	No	
ERR_FATAL	Receive	Transmit	No	Yes	No	
Locked Transaction Message						
Unlock Message	Transmit	Receive	Yes	No	No	
Slot Power Limit Message						
Set Slot Power Limit [2]	Transmit	Receive	No	Yes	No	In Root Port mode, through software. [2]
Vendor-defined Messages						
Vendor Defined Type 0	Transmit Receive	Transmit Receive	Yes	No	No	
Vendor Defined Type 1	Transmit Receive	Transmit Receive	Yes	No	No	

Table 8–1. Supported Message Types [(2)](#) **(Part 3 of 3)**

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Hot Plug Messages						
Attention_indicator On	Transmit	Receive	No	Yes	No	As per the recommendations in the *PCI Express Base Specification Revision 2.1*, these messages are not transmitted to the Application Layer.
Attention_Indicator Blink	Transmit	Receive	No	Yes	No	
Attention_indicator_ Off	Transmit	Receive	No	Yes	No	
Power_Indicator On	Transmit	Receive	No	Yes	No	
Power_Indicator Blink	Transmit	Receive	No	Yes	No	
Power_Indicator Off	Transmit	Receive	No	Yes	No	
Attention Button_Pressed (1)	Receive	Transmit	No	No	Yes	

Notes to Table 8–1:

(1) In Endpoint mode.

(2) In the *PCI Express Base Specification Revision 2.1*, this message is no longer mandatory after link training.

Transaction Layer Routing Rules

Transactions adhere to the following routing rules:

- In the receive direction (from the PCI Express link), memory and I/O requests that match the defined base address register (BAR) contents and vendor-defined messages with or without data route to the receive interface. The Application Layer logic processes the requests and generates the read completions, if needed.
- In Endpoint mode, received Type 0 Configuration requests from the PCI Express upstream port route to the internal Configuration Space and the Arria V Hard IP for PCI Express generates and transmits the completion.
- The Hard IP handles supported received message transactions (Power Management and Slot Power Limit) internally. The Endpoint also supports the Unlock and Type 1 Messages. The Root Port supports Interrupt, Type 1 and error Messages.
- Vendor-defined Type 0 Message TLPs are passed to the Application Layer.
- The Transaction Layer treats all other received transactions (including memory or I/O requests that do not match a defined BAR) as Unsupported Requests. The Transaction Layer sets the appropriate error bits and transmits a completion, if needed. These Unsupported Requests are not made visible to the Application Layer; the header and data is dropped.

- For memory read and write request with addresses below 4 GBytes, requestors must use the 32-bit format. The Transaction Layer interprets requests using the 64-bit format for addresses below 4 GBytes as an Unsupported Request and does not send them to the Application Layer. If Error Messaging is enabled, an error Message TLP is sent to the Root Port. Refer to "Errors Detected by the Transaction Layer" on page 12–3 for a comprehensive list of TLPs the Hard IP does not forward to the Application Layer.
- The Transaction Layer sends all memory and I/O requests, as well as completions generated by the Application Layer and passed to the transmit interface, to the PCI Express link.
- The Hard IP can generate and transmit power management, interrupt, and error signaling messages automatically under the control of dedicated signals. Additionally, it can generate MSI requests under the control of the dedicated signals.
- In Root Port mode, the Application Layer can issue Type 0 or Type 1 Configuration TLPs on the Avalon-ST TX bus.
    - The Type 1 Configuration TLPs are sent downstream on the PCI Express link toward the Endpoint that matches the Completer ID set in the transmit packet. If the bus number of the Type 1 Configuration TLP matches the Subordinate Bus Number register value in the Root Port Configuration Space, the TLP is converted to a Type 0 TLP.
    - The Type 0 Configuration TLPs are only routed to the Configuration Space of the Hard IP and are not sent downstream on the PCI Express link.

Receive Buffer Reordering

The RX datapath implements a RX buffer reordering function that allows posted and completion transactions to pass non-posted transactions (as allowed by PCI Express ordering rules) when the Application Layer is unable to accept additional non-posted transactions.

The Application Layer dynamically enables the RX buffer reordering by asserting the `rx_mask` signal. The `rx_mask` signal blocks non-posted request transactions made to the Application Layer interface so that only posted and completion transactions are presented to the Application Layer. Table 8–2 lists the transaction ordering rules.

Table 8–2. Transaction Ordering Rules (1)– (9) (Part 1 of 2)

Row Pass Column		Posted Request		Non Posted Request				Completion			
		Memory Write or Message Request		Read Request		I/O or Cfg Write Request		Read Completion		I/O or Cfg Write Completion	
		Spec (10)	Hard IP	Spec	Hard IP	Spec	Hard IP	Spec	Hard IP	Spec	Hard IP
Posted	Memory Write or Message Request	N (11) Y/N (12)	N (11) N (12)	Y	Y	Y	Y	Y/N (11) Y (12)	N (11) N (12)	Y/N (11) Y (12)	N (11) N (12)

Table 8–2. Transaction Ordering Rules (1)– (9) (Part 2 of 2)

NonPosted	Read Request	N	N	Y/N	N (11)	Y/N	N (12)	Y/N	N	Y/N	N
	I/O or Configuration Write Request	N	N	Y/N	N (13)	Y/N	N (14)	Y/N	N	Y/N	N
Completion	Read Completion	N (11) Y/N (12)	N (11) N (12)	Y	Y	Y	Y	Y/N (11) N (12)	N (11) N (12)	Y/N	N
	I/O or Configuration Write Completion	Y/N	N	Y	Y	Y	Y	Y/N	N	Y/N	N

Notes to Table 8–2:

(1) A Memory Write or Message Request with the Relaxed Ordering Attribute bit clear (b'0) must not pass any other Memory Write or Message Request.

(2) A Memory Write or Message Request with the Relaxed Ordering Attribute bit set (b'1) is permitted to pass any other Memory Write or Message Request.

(3) Endpoints, Switches, and Root Complex may allow Memory Write and Message Requests to pass Completions or be blocked by Completions.

(4) Memory Write and Message Requests can pass Completions traveling in the PCI Express to PCI directions to avoid deadlock.

(5) If the Relaxed Ordering attribute is not set, then a Read Completion cannot pass a previously enqueued Memory Write or Message Request.

(6) If the Relaxed Ordering attribute is set, then a Read Completion is permitted to pass a previously enqueued Memory Write or Message Request.

(7) Read Completion associated with different Read Requests are allowed to be blocked by or to pass each other.

(8) Read Completions for Request (same Transaction ID) must return in address order.

(9) Non-posted requests cannot pass other non-posted requests.

(10) Refers to the *PCI Express Base Specification 3.0.*

(11) `CfgRd0` can pass `IORd` or `MRd`.

(12) `CfgWr0` can `IORd` or `MRd`.

(13) `CfgRd0` can pass `IORd` or `MRd`.

(14) `CfrWr0` can pass `IOWr`.

MSI requests are conveyed in exactly the same manner as PCI Express memory write requests and are indistinguishable from them in terms of flow control, ordering, and data integrity.

9. Optional Features

This chapter provides information on several additional topics. It includes the following sections:

- ECRC
- Lane Initialization and Reversal

ECRC

ECRC ensures end-to-end data integrity for systems that require high reliability. You can specify this option under the **Error Reporting** heading. The ECRC function includes the ability to check and generate ECRC. In addition, the ECRC function can also forward the TLP with ECRC to the RX port of the Application Layer. When using ECRC forwarding mode, the ECRC check and generate are performed in the Application Layer.

You must turn on **Advanced error reporting (AER)**, **ECRC checking**, **ECRC generation**, and **ECRC forwarding** under the **PCI Express/PCI Capabilities** page of the parameter editor to enable this functionality.

For more information about error handling, refer to the *Error Signaling and Logging* which is Section 6.2 of the *PCI Express Base Specification, Rev. 2.1*.

ECRC on the RX Path

When the **ECRC generation** option is turned on, errors are detected when receiving TLPs with a bad ECRC. If the **ECRC generation** option is turned off, no error detection occurs. If the **ECRC forwarding** option is turned on, the ECRC value is forwarded to the Application Layer with the TLP. If the **ECRC forwarding** option is turned off, the ECRC value is not forwarded.

Table 9–1 summarizes the RX ECRC functionality for all possible conditions.

Table 9–1. ECRC Operation on RX Path (Part 1 of 2)

ECRC Forwarding	ECRC Check Enable (1)	ECRC Status	Error	TLP Forward to Application Layer
No	No	none	No	Forwarded
		good	No	Forwarded without its ECRC
		bad	No	Forwarded without its ECRC
	Yes	none	No	Forwarded
		good	No	Forwarded without its ECRC
		bad	Yes	Not forwarded

Table 9–1. ECRC Operation on RX Path (Part 2 of 2)

ECRC Forwarding	ECRC Check Enable (1)	ECRC Status	Error	TLP Forward to Application Layer
Yes	No	none	No	Forwarded
		good	No	Forwarded with its ECRC
		bad	No	Forwarded with its ECRC
	Yes	none	No	Forwarded
		good	No	Forwarded with its ECRC
		bad	Yes	Not forwarded

Note to Table 9–1:

(1) The `ECRC Check Enable` is in the `Configuration Space Advanced Error Capabilities and Control` Register.

ECRC on the TX Path

When the **ECRC generation** option is on, the TX path generates ECRC. If you turn on **ECRC forwarding**, the ECRC value is forwarded with the TLP. Table 9–2 summarizes the TX ECRC generation and forwarding. In this table, if `TD` is 1, the TLP includes an ECRC. `TD` is the TL digest bit of the TL packet described in Appendix A, Transaction Layer Packet (TLP) Header Formats.

Table 9–2. ECRC Generation and Forwarding on TX Path (1)

ECRC Forwarding	ECRC Generation Enable (2)	TLP on Application Layer	TLP on Link	Comments
No	No	TD=0, without ECRC	TD=0, without ECRC	
		TD=1, without ECRC	TD=0, without ECRC	
	Yes	TD=0, without ECRC	TD=1, with ECRC	ECRC is generated
		TD=1, without ECRC	TD=1, with ECRC	
Yes	No	TD=0, without ECRC	TD=0, without ECRC	Core forwards the ECRC
		TD=1, with ECRC	TD=1, with ECRC	
	Yes	TD=0, without ECRC	TD=0, without ECRC	
		TD=1, with ECRC	TD=1, with ECRC	

Notes to Table 9–2:

(1) All unspecified cases are unsupported and the behavior of the Hard IP is unknown.

(2) The `ECRC Generation Enable` is in the `Configuration Space Advanced Error Capabilities and Control` Register.

Lane Initialization and Reversal

Connected components that include IP blocks for PCI Express need not support the same number of lanes. The ×4 variations support initialization and operation with components that have 1, 2, or 4 lanes. The ×8 variant supports initialization and operation with components that have 1, 2, 4, or 8 lanes.

The Arria V Hard IP for PCI Express supports lane reversal, which permits the logical reversal of lane numbers for the ×1, ×2, ×4, and ×8 configurations. Lane reversal allows more flexibility in board layout, reducing the number of signals that must cross over each other when routing the PCB.

Table 9–3 summarizes the lane assignments for normal configuration.

Table 9–3. Lane Assignments without Lane Reversal

Lane Number	7	6	5	4	3	2	1	0
×8 IP core	7	6	5	4	3	2	1	0
×4 IP core	—	—	—	—	3	2	1	0
×1 IP core	—	—	—	—	—	—	—	0

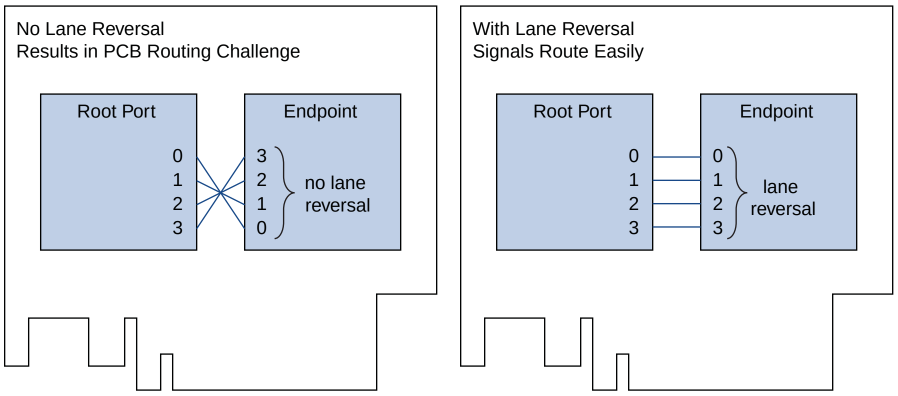
Table 9–4 summarizes the lane assignments with lane reversal.

Table 9–4. Lane Assignments with Lane Reversal

Core Config	8				4				1			
Slot Size	8	4	2	1	8	4	2	1	8	4	2	1
Lane assignments	7:0,6:1,5:2,4:3,3:4, 2:5,1:6,0:7	3:4,2:5, 1:6,0:7	1:6, 0:7	0:7	7:0,6:1, 5:2,4:3	3:0,2:1, 1:2,0:3	3:0, 2:1	3:0	7:0	3:0	1:0	0:0

Figure 9–1 illustrates a PCI Express card with ×4 IP Root Port and a ×4 Endpoint on the top side of the PCB. Connecting the lanes without lane reversal creates routing problems. Using lane reversal, solves the problem.

Figure 9–1. Using Lane Reversal to Solve PCB Routing Problems



10. Interrupts

This chapter describes interrupts for the following configurations:

- Interrupts for Endpoints
- Interrupts for Root Ports

Refer to "Interrupts for Endpoints" on page 5–23 and "Interrupts for Root Ports" on page 5–24 for descriptions of the interrupt signals.

Interrupts for Endpoints

The Arria V Hard IP for PCI Express provides support for PCI Express legacy interrupts, MSI, and MSI-X interrupts when configured in Endpoint mode. The MSI, MSI-X, and legacy interrupts are *mutually exclusive.* After power up, the Hard IP block starts in INTX mode, after which time software decides whether to switch to MSI mode by programming the `msi_enable` bit of the `MSI message control` register (bit[16] of 0x050) to 1 or to MSI-X mode if you turn on **Implement MSI-X** under the **PCI Express/PCI Capabilities** tab using the parameter editor. If you turn on the **Implement MSI-X** option, you should implement the MSI-X table structures at the memory space pointed to by the BARs.

Refer to section 6.1 of *PCI Express 2.1 Base Specification* for a general description of PCI Express interrupt support for Endpoints.

MSI Interrupts

MSI interrupts are signaled on the PCI Express link using a single dword memory write TLPs generated internally by the Arria V Hard IP for PCI Express. The `app_msi_req` input port controls MSI interrupt generation. When the input port asserts `app_msi_req`, it causes a MSI posted write TLP to be generated based on the MSI configuration register values and the `app_msi_tc` and `app_msi_num` input ports. Software uses configuration requests to program the MSI registers. To enable MSI interrupts, software must first set the `MSI enable` bit (Table 5–15 on page 5–33) and then disable legacy interrupts by setting the `Interrupt Disable` (Table 6–2 on page 6–2) bit.

Figure 10–1 illustrates the architecture of the MSI handler block.

Figure 10–1. MSI Handler Block

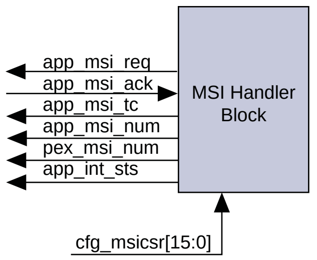
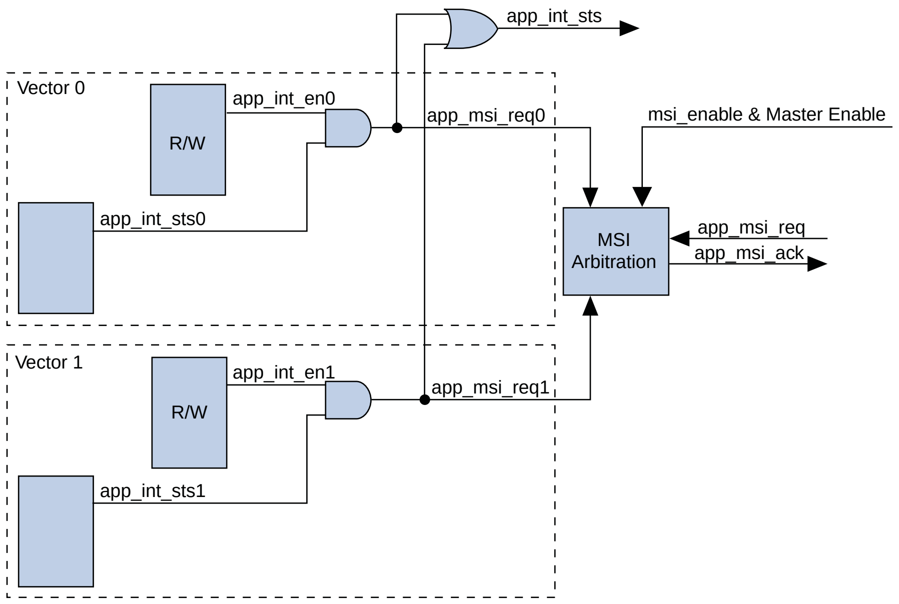


Figure 10–2 illustrates a possible implementation of the MSI handler block with a per vector enable bit. A global Application Layer interrupt enable can also be implemented instead of this per vector MSI.

Figure 10–2. Example Implementation of the MSI Handler Block



There are 32 possible MSI messages. The number of messages requested by a particular component does not necessarily correspond to the number of messages allocated. For example, in Figure 10–3, the Endpoint requests eight MSIs but is only allocated two. In this case, you must design the Application Layer to use only two allocated messages.

Figure 10–3. MSI Request Example

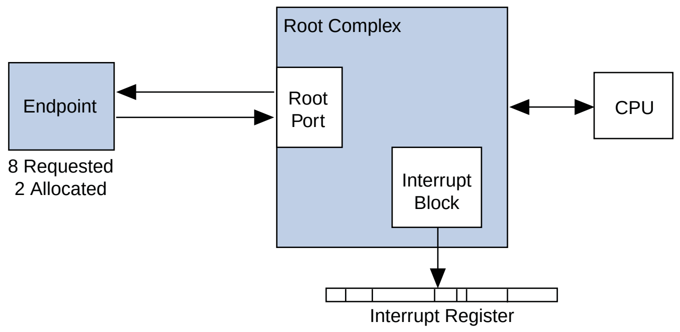
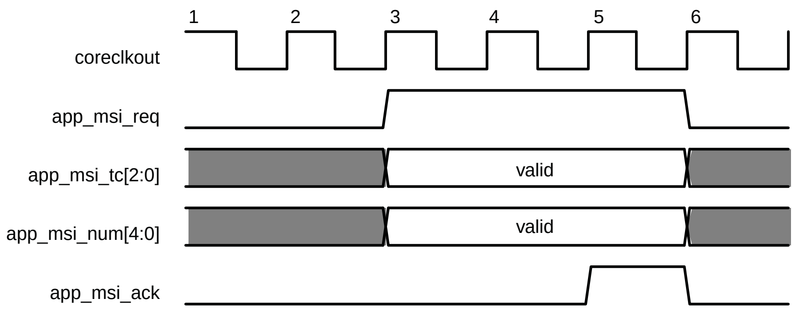


Figure 10–4 illustrates the interactions among MSI interrupt signals for the Root Port in Figure 10–3. The minimum latency possible between `app_msi_req` and `app_msi_ack` is one clock cycle.

Figure 10–4. MSI Interrupt Signals Waveform (1)



Note to Figure 10–4:

(1) `app_msi_req` can extend beyond `app_msi_ack` before deasserting. F

MSI-X

You can enable MSI-X interrupts by turning on **Implement MSI-X** on the **MSI-X** tab under the **PCI Express/PCI Capabilities** heading using the parameter editor. If you turn on the **Implement MSI-X** option, you should implement the MSI-X table structures at the memory space pointed to by the BARs as part of your Application Layer.

MSI-X TLPs are generated by the Application Layer and sent through the TX interface. They are single dword memory writes so that `Last DW Byte Enable` in the TLP header must be set to 4b'0000. MSI-X TLPs should be sent only when enabled by the MSI-X enable and the function mask bits in the message control for MSI-X Configuration register. These bits are available on the `tl_cfg_ctl` output bus.

For more information about implementing the MSI-X capability structure, refer Section 6.8.2. of the *PCI Local Bus Specification, Revision 3.0*.

Legacy Interrupts

Legacy interrupts are signaled on the PCI Express link using message TLPs that are generated internally by the Arria V Hard IP for PCI Express IP core. The `app_int_sts` input port controls interrupt generation. When the input port asserts `app_int_sts`, it causes an `Assert_INTA` message TLP to be generated and sent upstream. Deassertion of the `app_int_sts` input port causes a `Deassert_INTA` message TLP to be generated and sent upstream. To use legacy interrupts, you must clear the `Interrupt Disable` bit, which is bit 10 of the `Command` register (Table 6–2 on page 6–2). Then, turn off the `MSI Enable` bit (Table 5–15 on page 5–33.)

Figure 10–5 illustrates interrupt timing for the legacy interface. In this figure the assertion of `app_int_ack` instructs the Hard IP for PCI Express to send a `Assert_INTA` message TLP.

Figure 10–5. Legacy Interrupt Assertion

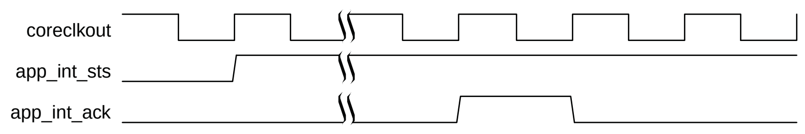


Figure 10–6 illustrates the timing for deassertion of legacy interrupts. The assertion of `app_int_ack` instructs the Hard IP for PCI Express to send a `Deassert_INTA` message.

Figure 10–6. Legacy Interrupt Deassertion

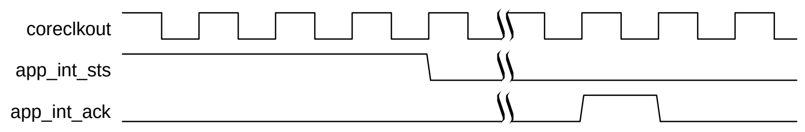


Table 10–1 describes 3 example implementations; 1 in which all 32 MSI messages are allocated and 2 in which only 4 are allocated.

Table 10–1. MSI Messages Requested, Allocated, and Mapped

MSI	Allocated		
	32	**4**	**4**
System error	31	3	3
Hot plug and power management event	30	2	3
Application Layer	29:0	1:0	2:0

MSI interrupts generated for Hot Plug, Power Management Events, and System Errors always use TC0. MSI interrupts generated by the Application Layer can use any Traffic Class. For example, a DMA that generates an MSI at the end of a transmission can use the same traffic control as was used to transfer data.

Interrupts for Root Ports

In Root Port mode, the Arria V Hard IP for PCI Express IP core receives interrupts through two different mechanisms:

- MSI—Root Ports receive MSI interrupts through the Avalon-ST RX TLP of type `MWr`. This is a memory mapped mechanism.
- Legacy—Legacy interrupts are translated into TLPs of type `Message Interrupt` which is sent to the Application Layer using the `int_status[3:0]` pins.

Normally, the Root Port services rather than sends interrupts; however, in two circumstances the Root Port can send an interrupt to itself to record error conditions:

- When the AER option is enabled, the `aer_msi_num[4:0]` signal indicates which MSI is being sent to the root complex when an error is logged in the AER Capability structure. This mechanism is an alternative to using the `serr_out` signal. The `aer_msi_num[4:0]` is only used for Root Ports and you must set it to a constant value. It cannot toggle during operation.
- If the Root Port detects a Power Management Event, the `pex_msi_num[4:0]` signal is used by Power Management or Hot Plug to determine the offset between the base message interrupt number and the message interrupt number to send through MSI. The user must set `pex_msi_num[4:0]`to a fixed value.

The `Root Error Status` register reports the status of error messages. The `Root Error Status` register is part of the PCI Express AER Extended Capability structure. It is located at offset 0x830 of the Configuration Space registers.

11. Flow Control

Throughput analysis requires that you understand the Flow Control Loop, shown in "Flow Control Update Loop" on page 11–2. This chapter discusses the Flow Control Loop and strategies to improve throughput. It covers the following topics:

- Throughput of Posted Writes
- Throughput of Non-Posted Reads

Throughput of Posted Writes

The throughput of posted writes is limited primarily by the Flow Control Update loop shown in Figure 11–1. If the write requester sources the data as quickly as possible, and the completer consumes the data as quickly as possible, then the Flow Control Update loop may be the biggest determining factor in write throughput, after the actual bandwidth of the link.

Figure 11–1 shows the main components of the Flow Control Update loop with two communicating PCI Express ports:

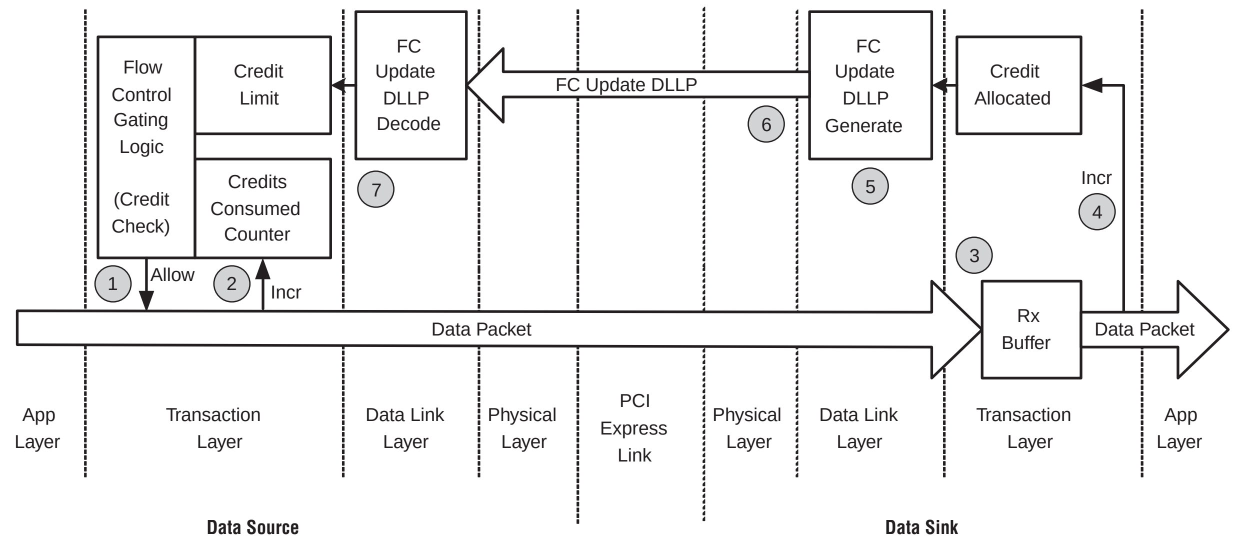
- Write Requester
- Write Completer

As the *PCI Express Base Specification 2.1* describes, each transmitter, the write requester in this case, maintains a `Credit Limit Register` and a `Credits Consumed Register`. The `Credit Limit Register` is the sum of all credits issued by the receiver, the write completer in this case. The `Credit Limit Register` is initialized during the flow control initialization phase of link initialization and then updated during operation by Flow Control (FC) Update DLLPs. The `Credits Consumed Register` is the sum of all credits consumed by packets transmitted. Separate `Credit Limit` and `Credits Consumed Registers` exist for each of the six types of Flow Control:

- Posted Headers
- Posted Data
- Non-Posted Headers
- Non-Posted Data
- Completion Headers
- Completion Data

Each receiver also maintains a credit allocated counter which is initialized to the total available space in the RX buffer (for the specific Flow Control class) and then incremented as packets are pulled out of the RX buffer by the Application Layer. The value of this register is sent as the FC Update DLLP value.

Figure 11–1. Flow Control Update Loop



The following numbered steps describe each step in the Flow Control Update loop. The corresponding numbers on Figure 11–1 show the general area to which they correspond.

1. When the Application Layer has a packet to transmit, the number of credits required is calculated. If the current value of the credit limit minus credits consumed is greater than or equal to the required credits, then the packet can be transmitted immediately. However, if the credit limit minus credits consumed is less than the required credits, then the packet must be held until the credit limit is increased to a sufficient value by an FC Update DLLP. This check is performed separately for the header and data credits; a single packet consumes only a single header credit.
2. After the packet is selected for transmission the `Credits Consumed Register` is incremented by the number of credits consumed by this packet. This increment happens for both the header and data `Credit Consumed Registers`.
3. The packet is received at the other end of the link and placed in the RX buffer.
4. At some point the packet is read out of the RX buffer by the Application Layer. After the entire packet is read out of the RX buffer, the `Credit Allocated Register` can be incremented by the number of credits the packet has used. There are separate `Credit Allocated Registers` for the header and data credits.
5. The value in the `Credit Allocated Registers` is used to create an FC Update DLLP.

6. After an FC Update DLLP is created, it arbitrates for access to the PCI Express link. The FC Update DLLPs are typically scheduled with a low priority; consequently, a continuous stream of Application Layer TLPs or other DLLPs (such as ACKs) can delay the FC Update DLLP for a long time. To prevent starving the attached transmitter, FC Update DLLPs are raised to a high priority under the following three circumstances:

   a. When the last sent credit allocated counter minus the amount of received data is less than maximum payload and the current credit allocated counter is greater than the last sent credit counter. Essentially, this means the data sink knows the data source has less than a full maximum payload worth of credits, and therefore is starving.

   b. When an internal timer expires from the time the last FC Update DLLP was sent, which is configured to 30 µs to meet the *PCI Express Base Specification* for resending FC Update DLLPs.

   c. When the credit allocated counter minus the last sent credit allocated counter is greater than or equal to 25% of the total credits available in the RX buffer, then the FC Update DLLP request is raised to high priority.

   After arbitrating, the FC Update DLLP that won the arbitration to be the next item is transmitted. In the worst case, the FC Update DLLP may need to wait for a maximum sized TLP that is currently being transmitted to complete before it can be sent.

7. The FC Update DLLP is received back at the original write requester and the credit limit value is updated. If packets are stalled waiting for credits, they can now be transmitted.

To allow the write requester to transmit packets continuously, the `credit allocated` and the `credit limit` counters must be initialized with sufficient credits to allow multiple TLPs to be transmitted while waiting for the FC Update DLLP that corresponds to the freeing of credits from the very first TLP transmitted.

You can use the **RX Buffer space allocation - Desired performance for received requests** to configure the RX buffer with enough space to meet the credit requirements of your system.

Throughput of Non-Posted Reads

To support a high throughput for read data, you must analyze the overall delay from the time the Application Layer issues the read request until all of the completion data is returned. The Application Layer must be able to issue enough read requests, and the read completer must be capable of processing these read requests quickly enough (or at least offering enough non-posted header credits) to cover this delay.

However, much of the delay encountered in this loop is well outside the Arria V Hard IP for PCI Express and is very difficult to estimate. PCI Express switches can be inserted in this loop, which makes determining a bound on the delay more difficult.

Nevertheless, maintaining maximum throughput of completion data packets is important. Endpoints must offer an infinite number of completion credits. Endpoints must buffer this data in the RX buffer until the Application Layer can process it. Because the Endpoint is no longer managing the RX buffer through the flow control mechanism, the Application Layer must manage the RX buffer by the rate at which it issues read requests.

To determine the appropriate settings for the amount of space to reserve for completions in the RX buffer, you must make an assumption about the length of time until read completions are returned. This assumption can be estimated in terms of an additional delay, beyond the FC Update Loop Delay, as discussed in the section "Throughput of Posted Writes" on page 11–1. The paths for the read requests and the completions are not exactly the same as those for the posted writes and FC Updates in the PCI Express logic. However, the delay differences are probably small compared with the inaccuracy in the estimate of the external read to completion delays.

With multiple completions, the number of available credits for completion headers must be larger than the completion data space divided by the maximum packet size. Instead, the credit space for headers must be the completion data space (in bytes) divided by 64, because this is the smallest possible read completion boundary. Setting the **RX Buffer space allocation – Desired performance for received completions** to **High** under the **System Settings** heading when specifying parameter settings configures the RX buffer with enough space to meet this requirement. You can adjust this setting up or down from the **High** setting to tailor the RX buffer size to your delays and required performance.

You can also control the maximum amount of outstanding read request data. This amount is limited by the number of header tag values that can be issued by the Application Layer and by the maximum read request size that can be issued. The number of header tag values that can be in use is also limited by the Arria V Hard IP for PCI Express. You can specify 32 or 64 tags though configuration software to restrict the Application Layer to use only 32 tags. In commercial PC systems, 32 tags are usually sufficient to maintain optimal read throughput.

12. Error Handling

Each PCI Express compliant device must implement a basic level of error management and can optionally implement advanced error management. The Altera Arria V Hard IP for PCI Express implements both basic and advanced error reporting. Given its position and role within the fabric, error handling for a Root Port is more complex than that of an Endpoint.

The *PCI Express Base Specification 2.1* defines three types of errors, outlined in Table 12–1.

Table 12–1. Error Classification

Type	Responsible Agent	Description
Correctable	Hardware	While correctable errors may affect system performance, data integrity is maintained.
Uncorrectable, non-fatal	Device software	Uncorrectable, non-fatal errors are defined as errors in which data is lost, but system integrity is maintained. For example, the fabric may lose a particular TLP, but it still works without problems.
Uncorrectable, fatal	System software	Errors generated by a loss of data and system failure are considered uncorrectable and fatal. Software must determine how to handle such errors: whether to reset the link or implement other means to minimize the problem.

The following sections describe the errors detected by the three layers of the PCI Express protocol and error logging. It includes the following sections:

- Physical Layer Errors
- Data Link Layer Errors
- Transaction Layer Errors
- Error Reporting and Data Poisoning
- Uncorrectable and Correctable Error Status Bits

Physical Layer Errors

Table 12–2 describes errors detected by the Physical Layer.

Table 12–2. Errors Detected by the Physical Layer [(1)]

Error	Type	Description
Receive port error	Correctable	This error has the following 3 potential causes: ■ Physical coding sublayer error when a lane is in L0 state. These errors are reported to the Hard IP block via the per lane PIPE interface input receive status signals, `rxstatus<`*`lane_number`*`>[2:0]` using the following encodings: 100: 8B/10B Decode Error 101: Elastic Buffer Overflow 110: Elastic Buffer Underflow 111: Disparity Error ■ Deskew error caused by overflow of the multilane deskew FIFO. ■ Control symbol received in wrong lane.

Note to Table 12–2:

(1) Considered optional by the PCI Express specification.

Data Link Layer Errors

Table 12–3 describes errors detected by the Data Link Layer.

Table 12–3. Errors Detected by the Data Link Layer

Error	Type	Description
Bad TLP	Correctable	This error occurs when a LCRC verification fails or when a sequence number error occurs.
Bad DLLP	Correctable	This error occurs when a CRC verification fails.
Replay timer	Correctable	This error occurs when the replay timer times out.
Replay num rollover	Correctable	This error occurs when the replay number rolls over.
Data Link Layer protocol	Uncorrectable (fatal)	This error occurs when a sequence number specified by the Ack/Nak block in the Data Link Layer (`AckNak_Seq_Num`) does not correspond to an unacknowledged TLP. (Refer to "Data Link Layer" on page 4–7.)

Transaction Layer Errors

Table 12–4 describes errors detected by the Transaction Layer.

Table 12–4. Errors Detected by the Transaction Layer (Part 1 of 3)

Error	Type	Description
Poisoned TLP received	Uncorrectable (non-fatal)	This error occurs if a received Transaction Layer packet has the EP poison bit set. The received TLP is passed to the Application Layer and the Application Layer logic must take appropriate action in response to the poisoned TLP. Refer to "2.7.2.2 Rules for Use of Data Poisoning" in the *PCI Express Base Specification 2.1* for more information about poisoned TLPs.
ECRC check failed (1)	Uncorrectable (non-fatal)	This error is caused by an ECRC check failing despite the fact that the TLP is not malformed and the LCRC check is valid. The Hard IP block handles this TLP automatically. If the TLP is a non-posted request, the Hard IP block generates a completion with completer abort status. In all cases the TLP is deleted in the Hard IP block and not presented to the Application Layer.
Unsupported Request for Endpoints	Uncorrectable (non-fatal)	This error occurs whenever a component receives any of the following Unsupported Requests: ■ Type 0 Configuration Requests for a non-existing function. ■ Completion transaction for which the Requester ID does not match the bus/device. ■ Unsupported message. ■ A Type 1 Configuration Request TLP for the TLP from the PCIe link. ■ A locked memory read (MEMRDLK) on Native Endpoint. ■ A locked completion transaction. ■ A 64-bit memory transaction in which the 32 MSBs of an address are set to 0. ■ A memory or I/O transaction for which there is no matching BAR. ■ A memory transaction when the Memory Space Enable bit (bit [1] of the PCI Command register at Configuration Space offset 0x4) is set to 0. ■ A poisoned configuration write request (`CfgWr0`) If the TLP is a non-posted request, the Hard IP block generates a completion with Unsupported Request status. In all cases the TLP is deleted in the Hard IP block and not presented to the Application Layer.
Unsupported Requests for Root Port	Uncorrectable fatal	This error occurs whenever a component receives an Unsupported Request including: ■ Unsupported message ■ A Type 0 Configuration Request TLP ■ A 64-bit memory transaction which the 32 MSBs of an address are set to 0. ■ A memory transaction that does not match a Windows address

Table 12–4. Errors Detected by the Transaction Layer (Part 2 of 3)

Error	Type	Description
Completion timeout	Uncorrectable (non-fatal)	This error occurs when a request originating from the Application Layer does not generate a corresponding completion TLP within the established time. It is the responsibility of the Application Layer logic to provide the completion timeout mechanism. The completion timeout should be reported from the Transaction Layer using the `cpl_err[0]` signal.
Completer abort [(1)]	Uncorrectable (non-fatal)	The Application Layer reports this error using the `cpl_err[2]`signal when it aborts receipt of a TLP.
Unexpected completion	Uncorrectable (non-fatal)	This error is caused by an unexpected completion transaction. The Hard IP block handles the following conditions: ■ The Requester ID in the completion packet does not match the Configured ID of the Endpoint. ■ The completion packet has an invalid tag number. (Typically, the tag used in the completion packet exceeds the number of tags specified.) ■ The completion packet has a tag that does not match an outstanding request. ■ The completion packet for a request that was to I/O or Configuration Space has a length greater than 1 dword. ■ The completion status is Configuration Retry Status (CRS) in response to a request that was not to Configuration Space. In all of the above cases, the TLP is not presented to the Application Layer; the Hard IP block deletes it. The Application Layer can detect and report other unexpected completion conditions using the `cpl_err[2]` signal. For example, the Application Layer can report cases where the total length of the received successful completions do not match the original read request length.
Receiver overflow [(1)]	Uncorrectable (fatal)	This error occurs when a component receives a TLP that violates the FC credits allocated for this type of TLP. In all cases the hard IP block deletes the TLP and it is not presented to the Application Layer.
Flow control protocol error (FCPE) [(1)]	Uncorrectable (fatal)	A receiver must never cumulatively issue more than 2047 outstanding unused data credits or 127 header credits to the transmitter. If Infinite credits are advertised for a particular TLP type (posted, non-posted, completions) during initialization, update FC DLLPs must continue to transmit infinite credits for that TLP type.
Malformed TLP	Uncorrectable (fatal)	This error is caused by any of the following conditions: ■ The data payload of a received TLP exceeds the maximum payload size. ■ The `TD` field is asserted but no TLP digest exists, or a TLP digest exists but the `TD` bit of the PCI Express request header packet is not asserted. ■ A TLP violates a byte enable rule. The Hard IP block checks for this violation, which is considered optional by the PCI Express specifications. ■ A TLP in which the `type` and `length` fields do not correspond with the total length of the TLP. ■ A TLP in which the combination of format and type is not specified by the PCI Express specification.

Table 12–4. Errors Detected by the Transaction Layer (Part 3 of 3)

Error	Type	Description
Malformed TLP (continued)	Uncorrectable (fatal)	■ A request specifies an address/length combination that causes a memory space access to exceed a 4 KByte boundary. The Hard IP block checks for this violation, which is considered optional by the PCI Express specification. ■ Messages, such as Assert_INTX, Power Management, Error Signaling, Unlock, and Set Power Slot Limit, must be transmitted across the default traffic class. The Hard IP block deletes the malformed TLP; it is not presented to the Application Layer.

Note to Table 12–4:

(1) Considered optional by the *PCI Express Base Specification Revision 2.1*.

Error Reporting and Data Poisoning

How the Endpoint handles a particular error depends on the configuration registers of the device.

Refer to the *PCI Express Base Specification 2.1* for a description of the device signaling and logging for an Endpoint.

The Hard IP block implements data poisoning, a mechanism for indicating that the data associated with a transaction is corrupted. Poisoned TLPs have the error/poisoned bit of the header set to 1 and observe the following rules:

- Received poisoned TLPs are sent to the Application Layer and status bits are automatically updated in the Configuration Space.
- Received poisoned Configuration Write TLPs are not written in the Configuration Space.
- The Configuration Space never generates a poisoned TLP; the error/poisoned bit of the header is always set to 0.

Poisoned TLPs can also set the parity error bits in the PCI Configuration Space Status register. Table 12–5 lists the conditions that cause parity errors.

Table 12–5. Parity Error Conditions

Status Bit	Conditions
Detected parity error (status register bit 15)	Set when any received TLP is poisoned.
Master data parity error (status register bit 8)	This bit is set when the command register parity enable bit is set and one of the following conditions is true: ■ The poisoned bit is set during the transmission of a Write Request TLP. ■ The poisoned bit is set on a received completion TLP.

Poisoned packets received by the Hard IP block are passed to the Application Layer. Poisoned transmit TLPs are similarly sent to the link.

Uncorrectable and Correctable Error Status Bits

The following section is reprinted with the permission of PCI-SIG. Copyright 2010 PCI-SIGR.

Figure 12–1 illustrates the Uncorrectable Error Status register. The default value of all the bits of this register is 0. An error status bit that is set indicates that the error condition it represents has been detected. Software may clear the error status by writing a 1 to the appropriate bit.

Figure 12–1. Uncorrectable Error Status Register

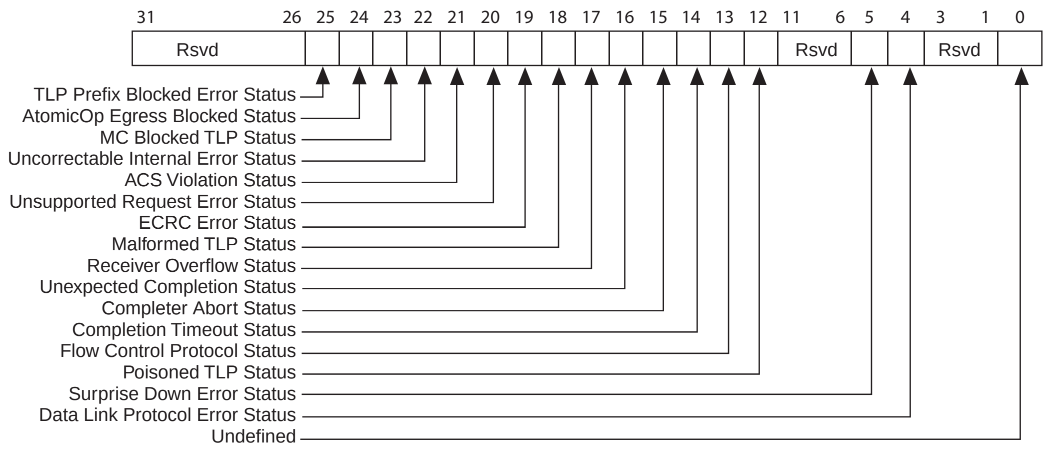
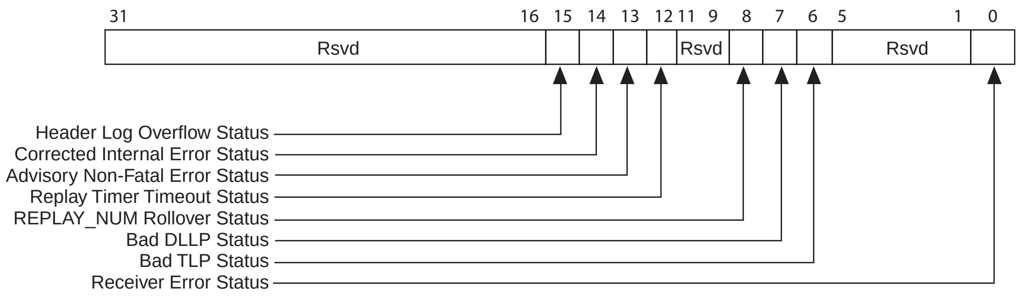

31
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
6
5
4
3
1
0
Rsvd
Rsvd
Rsvd
TLP Prefix Blocked Error Status
AtomicOp Egress Blocked Status
MC Blocked TLP Status
Uncorrectable Internal Error Status
ACS Violation Status
Unsupported Request Error Status
ECRC Error Status
Malformed TLP Status
Receiver Overflow Status
Unexpected Completion Status
Completer Abort Status
Completion Timeout Status
Flow Control Protocol Status
Poisoned TLP Status
Surprise Down Error Status
Data Link Protocol Error Status
Undefined

Figure 12–2 illustrates the Correctable Error Status register. The default value of all the bits of this register is 0. An error status bit that is set indicates that the error condition it represents has been detected. Software may clear the error status by writing a 1 to the appropriate bit.0

Figure 12–2. Correctable Error Status Register


31
16
15
14
13
12
11
9
8
7
6
5
1
0
Rsvd
Rsvd
Rsvd
Header Log Overflow Status
Corrected Internal Error Status
Advisory Non-Fatal Error Status
Replay Timer Timeout Status
REPLAY_NUM Rollover Status
Bad DLLP Status
Bad TLP Status
Receiver Error Status

13. Debugging

As you bring up your PCI Express system, you may face a number of issues related to FPGA configuration, link training, BIOS enumeration, data transfer, and so on. This chapter suggests some strategies to resolve the common issues that occur during hardware bring-up.

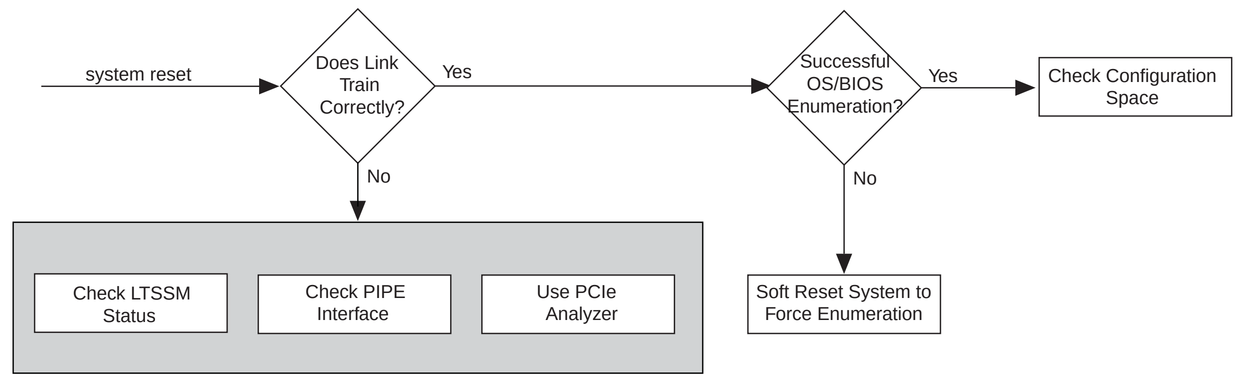
Hardware Bring-Up Issues

Typically, PCI Express hardware bring-up involves the following steps:

1. System reset
2. Linking training
3. BIOS enumeration

The following sections, describe how to debug the hardware bring-up flow. Altera recommends a systematic approach to diagnosing bring-up issues as illustrated in Figure 13–1.

Figure 13–1. Debugging Link Training Issues


system reset
Does Link Train Correctly?
Yes
No
Successful OS/BIOS Enumeration?
Yes
No
Check Configuration Space
Check LTSSM Status
Check PIPE Interface
Use PCIe Analyzer
Soft Reset System to Force Enumeration

Link Training

The Physical Layer automatically performs link training and initialization without software intervention. This is a well-defined process to configure and initialize the device's Physical Layer and link so that PCIe packets can be transmitted. If you encounter link training issues, viewing the actual data in hardware should help you determine the root cause. You can use the following tools to provide hardware visibility:

- SignalTap® II Embedded Logic Analyzer
- Third-party PCIe analyzer

Debugging Link Training Issues Using Quartus II SignalTap II

You can use SignalTap II Embedded Logic Analyzer to diagnose the LTSSM state transitions that are occurring and the PIPE interface.

Check Link Training and Status State Machine (ltssmstate[4:0])

The `ltssmstate[4:0]` bus encodes the status of LTSSM. The LTSSM state machine reflects the Physical Layer's progress through the link training process. For a complete description of the states these signals encode, refer to "Reset and Link Training Signals" on page 5–20. When link training completes successfully and the link is up, the LTSSM should remain stable in the L0 state.

When link issues occur, you can monitor `ltssmstate[4:0]` to determine whether link training fails before reaching the L0 state or the link was initially established (L0), but then lost due to an additional link training issue. If you have link training issues, you can check the actual link status in hardware using the SignalTap II logic analyzer. The LTSSM encodings indicate the LTSSM state of the Physical Layer as it proceeds through the link training process.

For more information about link training, refer to the "Link Training and Status State Machine (LTSSM) Descriptions" section of *PCI Express Base Specification 2.0*.

For more information about SignalTap, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Check PIPE Interface

Because the LTSSM signals reflect the behavior of one side of the PCI Express link, you may find it difficult to determine the root cause of the link issue solely by monitoring these signals. Monitoring the PIPE interface signals in addition to the `ltssmstate` bus provides greater visibility.

The PIPE interface is specified by Intel. This interface defines the MAC/PCS functional partitioning and defines the interface signals for these two sublayers. Using the SignalTap logic analyzer to monitor the PIPE interface signals provides more information about the devices that form the link.

During link training and initialization, different pre-defined Physical Layer Packets (PLPs), known as ordered sets are exchanged between the two devices on all lanes. All of these ordered sets have special symbols (K codes) that carry important information to allow two connected devices to exchange capabilities, such as link width, link data rate, lane reversal, lane-to-lane de-skew, and so on. You can track the ordered sets in the link initialization and training on both sides of the link to help you diagnose link issues. You can use SignalTap logic analyzer to determine the behavior. The following signals are some of the most important for diagnosing bring-up issues:

- `txdata<n>[15:0]/txdatak<n>[1:0]`—These signals show the data and control being transmitted from the Arria V Hard IP for PCI Express to the other device.
- `rxdata<n>[15:0]/rxdatak<n>[1:0]`—These signals show the data and control received by Hard P block from the other device.
- `phystatus<n>`—This signal communicates completion of several PHY requests.
- `rxstatus<n>[2:0]`—This signal encodes receive status and error codes for the receive data stream and receiver detection.

You can monitor the PIPE signals through the `test_out` bus.

The *PHY Interface for PCI Express Architecture* specification is available on the Intel website (**www.intel.com**).

Use Third-Party PCIe Analyzer

A third-party logic analyzer for PCI Express records the traffic on the physical link and decodes traffic, saving you the trouble of translating the symbols yourself. A third-party s logic analyzer can show the two-way traffic at different levels for different requirements. For high-level diagnostics, the analyzer shows the LTSSM flows for devices on both side of the link side-by-side. This display can help you see the link training handshake behavior and identify where the traffic gets stuck. A traffic analyzer can display the contents of packets so that you can verify the contents. For complete details, refer to the third-party documentation.

BIOS Enumeration Issues

Both FPGA programming (configuration) and the initialization of a PCIe link require time. There is some possibility that Altera FPGA including a Hard IP block for PCI Express may not be ready when the OS/BIOS begins enumeration of the device tree. If the FPGA is not fully programmed when the OS/BIOS begins its enumeration, the OS does not include the Hard IP for PCI Express in its device map. To eliminate this issue, you can do a soft reset of the system to retain the FPGA programming while forcing the OS/BIOS to repeat its enumeration.

A. Transaction Layer Packet (TLP) Header Formats

TLP Packet Format without Data Payload

Table A–1 through A–2 show the header format for TLPs without a data payload.

Table A–1. Memory Read Request, 32-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	0	0	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[31:2]																														0	0
Byte 12	Reserved																															

Table A–2. Memory Read Request, Locked 32-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	0	0	1	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[31:2]																														0	0
Byte 12	Reserved																															

Table A–3. Memory Read Request, 64-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	1	0	0	0	0	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[63:32]																															
Byte 12	Address[31:2]																														0	0

Table A–4. Memory Read Request, Locked 64-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	1	0	0	0	0	1	0	TC			0	0	0	0	T	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[63:32]																															
Byte 12	Address[31:2]																														0	0

Table A–5. Configuration Read Request Root Port (Type 1)

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	TD	EP	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Byte 4	Requester ID																Tag								0	0	0	0	First BE			
Byte 8	Bus Number								Device No					Func			0	0	0	0	Ext Reg				Register No						0	0
Byte 12	Reserved																															

Table A–6. I/O Read Request

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	TD	EP	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Byte 4	Requester ID																Tag								0	0	0	0	First BE			
Byte 8	Address[31:2]																														0	0
Byte 12	Reserved																															

Table A–7. Message without Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	1	1	0	r 2	r 1	r 0	0	TC			0	0	0	0	TD	EP	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Byte 4	Requester ID																Tag								Message Code							
Byte 8	Vendor defined or all zeros																															
Byte 12	Vendor defined or all zeros																															

Notes to Table A–7:

(1) Not supported in Avalon-MM.

Table A–8. Completion without Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	1	0	1	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Completer ID																Status			B	Byte Count											
Byte 8	Requester ID																Tag								0	Lower Address						
Byte 12	Reserved																															

Table A–9. Completion Locked without Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	1	0	1	1	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Completer ID																Status			B	Byte Count											

Table A–9. Completion Locked without Data

Byte 8	Requester ID	Tag	0	Lower Address
Byte 12	Reserved			

TLP Packet Format with Data Payload

Table A–10 through A–4 show the content for TLPs with a data payload.

Table A–10. Memory Write Request, 32-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	0	0	0	0	0	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[31:2]																														0	0
Byte 12	Reserved																															

Table A–11. Memory Write Request, 64-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	1	0	0	0	0	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[63:32]																															
Byte 12	Address[31:2]																														0	0

Table A–12. Configuration Write Request Root Port (Type 1)

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	TD	EP	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Byte 4	Requester ID																Tag								0	0	0	0	First BE			
Byte 8	Bus Number								Device No								0	0	0	0	Ext Reg				Register No						0	0
Byte 12	Reserved																															

Table A–13. I/O Write Request

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	TD	EP	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Byte 4	Requester ID																Tag								0	0	0	0	First BE			
Byte 8	Address[31:2]																														0	0
Byte 12	Reserved																															

Table A–14. Completion with Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	0	0	1	0	1	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Completer ID																Status			B	Byte Count											
Byte 8	Requester ID																Tag								0	Lower Address						
Byte 12	Reserved																															

Table A–15. Completion Locked with Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	0	0	1	0	1	1	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Completer ID																Status			B	Byte Count											
Byte 8	Requester ID																Tag								0	Lower Address						
Byte 12	Reserved																															

Table A–16. Message with Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	1	1	0	r2	r1	r0	0	TC			0	0	0	0	TD	EP	0	0	0	0	Length									
Byte 4	Requester ID																Tag								Message Code							
Byte 8	Vendor defined or all zeros for Slot Power Limit																															
Byte 12	Vendor defined or all zeros for Slots Power Limit																															

Additional Information

This chapter provides additional information about the document and Altera.

Revision History

The table below displays the revision history for the chapters in this User Guide.

Date	Version	Changes Made
November 2011	11.1	First release.

How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

Contact (1)	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Nontechnical support (general)	Email	nacomp@altera.com
(software licensing)	Email	authorization@altera.com

Note to Table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, $n + 1$. Variable names are enclosed in angle brackets (< >). For example, <*file name*> and <*project name*>**.pof** file.

Visual Cue	Meaning
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections in a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix n denotes an active-low signal. For example, `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
☞	The hand points to information that requires special attention.
?	The question mark directs you to a software help system with related information.
👣	The feet direct you to another document or website with related information.
🎥	The multimedia icon directs you to a related multimedia presentation.
⚠ CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
⚡ WARNING	A warning calls attention to a condition or possible situation that can cause you injury.
✉	The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.