Arria 10 Hard IP for PCI Express

User Guide for the Avalon Memory-Mapped Interface

Last updated for Altera Complete Design Suite: 13.1 Arria 10
UG-01145_avmm
December 2013

Subscribe
Send Feedback

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

Contents

Arria 10 Datasheet

1

December 2013

UG-01145_avmm

Subscribe Send Feedback

Altera® Arria 10 FPGAs include a configurable, hardened protocol stack for PCI Express that is compliant with *PCI Express® Base Specification 3.0*. The Hard IP for PCI Express PCIe IP core provides a choice of the following three interfaces:

- Avalon® Streaming (Avalon-ST)—This it the *native* interface to the PCIe Protocol stack's Transaction Layer. The Avalon-ST interface is the most flexible interface, but also requires a thorough understanding of the PCIe® Protocol.
- Avalon Memory-Mapped (Avalon-MM)—This interface is available as a bridge implemented in soft logic in the FPGA. It removes some of the complexities associated with the PCIe protocol. For example, it handles all of the Transaction Layer Protocol (TLP) encoding and decoding. Consequently, you can complete your design more quickly. The Avalon-MM is available in Qsys. Qsys is easy to understand and use.
- Avalon-MM with DMA—This interface is available as a bridge implemented in soft logic in the FPGA. This interface also handles TLP encoding and decoding. In addition, it includes DMA Read and DMA Write engines for the Avalon-MM interface. The Avalon-MM interface with DMA is currently only available for the 256-bit Gen3 configuration. The Quartus® II release 14.0 will include an Avalon-MM interface with DMA for the 128-bit configuration. If you have already architected your own DMA system with the Avalon-MM interface, you may want to continue to use it. However, you will probably benefit from the simplicity of having the DMA engines already implemented. For new users, Altera recommends starting with Avalon-MM interface with DMA. Avalon-MM interface with DMA is available in Qsys.

Refer to **Creating a System With Qsys** for more information about Qsys.

The following table shows the aggregate bandwidth of a PCI Express link for Gen1, Gen2, and Gen3 for 1, 4, and 8 lanes. The protocol specifies 2.5 giga-transfers per second for Gen1, 5 giga-transfers per second for Gen2, and 8.0 giga-transfers per second for Gen3. The following table provides bandwidths for a single transmit (TX) or receive (RX) channel, so that the numbers double for duplex operation. Gen1 and Gen2 use 8B/10B encoding which introduces a 20% overhead. In contrast, Gen3 uses 128b/130b encoding which reduces the data throughput lost to encoding to less than 1%.

Table 1-1: PCI Express Data Throughput

	Link Width		
	×1	×4	×8
PCI Express Gen1 (2.5 Gbps)	2	8	16

	Link Width		
	×1	×4	×8
PCI Express Gen2 (5.0 Gbps)	4	16	32
PCI Express Gen3 (8.0 Gbps)	7.87	31.51	63

Refer to the *PCI Express Reference Design for Stratix V Devices* for more information about calculating bandwidth for the hard IP implementation of PCI Express in many Altera FPGAs, including the Arria 10 Hard IP for PCI Express IP core.

Related Information

- **PCI Express Base Specification 3.0**
- **PCI Express DMA Reference Design for Stratix V Devices**

Features

The Arria 10 Hard IP for PCI Express supports the following features:

- Complete protocol stack including the Transaction, Data Link, and Physical Layers implemented as hard IP.
- Feature rich:
  - Support for ×1, ×4, and ×8 configurations with Gen1, Gen2, or Gen3 lane rates for Root Ports and Endpoints.
  - Dedicated 16 KByte receive buffer.
  - Dedicated hard reset controller.
  - Support for 256-bit Avalon-MM interface to Application Layer with embedded DMA capable of Gen3 ×8 data rate.
  - Optional support for Configuration via Protocol (CvP) using the PCIe link allowing the I/O and core bitstreams to be stored separately.
  - Qsys support using the Avalon Streaming (Avalon-ST) or Avalon Memory-Mapped (Avalon-MM) interface.
  - Support for 32- or 64-bit addressing for the Avalon-MM interface to the Application Layer.
  - Qsys example designs demonstrating parameterization, design modules and connectivity.
  - Extended credit allocation settings to better optimize the RX buffer space based on application type.
  - Support for multiple packets per cycle with the 256-bit Avalon-ST interface.
  - Optional end-to-end cyclic redundancy code (ECRC) generation and checking and advanced error reporting (AER) for high reliability applications.
- Easy to use:
  - Easy parameterization.
  - Substantial on-chip resource savings and guaranteed timing closure.
  - Easy adoption with no license requirement.
  - Example designs to get started.

Table 1-2: Hard IP for PCI Express Features

Feature	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM with Embedded DMA
MegaCore License	Free	Free	Free
Native Endpoint	Supported	Supported	Supported
Legacy Endpoint (2)	Supported	Not Supported	Not Supported
Root port	Supported	Supported	Not Supported
Gen1	×1, ×4, ×8	×1, ×4, ×8	Not Supported
Gen2	×1, ×4, ×8	×1, ×4, ×8	×8
Gen3	×1, ×4, ×8	×1, ×4	×4, ×8
MegaWizard Plug-In Manager design flow	Supported	Not supported	Not supported
Qsys design flow	Supported	Supported	Supported
64-bit Application Layer interface	Supported	Supported	Not supported
128-bit Application Layer interface	Supported	Supported	Not supported
256-bit Application Layer interface	Supported	Not Supported	Supported

Feature	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM with Embedded DMA
Transaction Layer Packet type (TLP)	• Memory Read Request • Memory Read Request-Locked • Memory Write Request • I/O Read Request • I/O Write Request • Configuration Read Request (Root Port) • Configuration Write Request (Root Port) • Message Request • Message Request with Data Payload • Completion without Data • Completion with Data • Completion for Locked Read without Data	• Memory Read Request • Memory Write Request • I/O Read Request—Root Port only • I/O Write Request—Root Port only • Configuration Read Request (Root Port) • Configuration Write Request (Root Port) • Completion without Data • Completion with Data • Memory Read Request (single dword) • Memory Write Request (single dword)	• Memory Read Request • Memory Write Request • Completion without Data • Completion with Data
Payload size	128–2048 bytes	128–256 bytes	128, 256, 512 bytes
Number of tags supported for non-posted requests	32 or 64	8	16
Out-of-order completions (transparent to the Application Layer)	Not supported	Supported	Supported
Requests that cross 4 KByte address boundary (transparent to the Application Layer)	Not supported	Supported	Supported
Polarity Inversion of PIPE interface signals	Supported	Supported	Supported
Number of MSI requests	16 or 32	1, 2, 4, 8, 16, or 32	1, 2, 4, 8, 16, or 32
MSI-X	Supported	Supported	Supported
Legacy interrupts	Supported	Supported	Supported
Expansion ROM	Supported	Not supported	Not supported

The purpose of the *Arria 10 Hard IP for PCI Express User Guide* is to explain how to use the Arria 10 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 *PCI Express Base Specification*.

Note: This release provides separate user guides for the three interfaces to the Application Layer. The *Related Information* provide links to all three versions.

Related Information

- **Arria 10 Hard IP for PCI Express User Guide for the Avalon Memory-Mapped Interface**
- **Arria 10 Hard IP for PCI Express User Guide for the Avalon Memory-Mapped Interface with DMA**
- **Arria 10 Hard IP for PCI Express User Guide for the Avalon Streaming Interface**

Release Information

The following tables provides information about this release of the Hard IP for PCI Express.

Table 1-3: Hard IP for PCI Express Release Information

Item	Description
Version	13.1 A10
Release Date	December 2013
Ordering Codes	No ordering code is required
Product IDs	There are no encrypted files for the Arria 10 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	

Device Family Support

The following table shows the level of support offered by the Arria 10 Hard IP for PCI Express.

Table 1-4: Device Family Support

Device Family	Support
Arria 10	Preliminary. The IP core is verified with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.

Device Family	Support
Other device families	Refer to the *Related Information* below for other device families:

Related Information

- **Arria V Hard IP for PCI Express User Guide**
- **Arria V GZ Hard IP for PCI Express User Guide for the Avalon Memory-Mapped Interface**
- **Arria V GZ Hard IP for PCI Express User Guide for the Avalon Memory-Mapped Interface with DMA**
- **Arria V GZ Hard IP for PCI Express User Guide for the Avalon Streaming Interface**
- **Cyclone V Hard IP for PCI Express User Guide**
- **IP Compiler for PCI Express User Guide**
- **Stratix V Hard IP for PCI Express User Guide for the Memory-Mapped Interface**
- **Stratix V Hard IP for PCI Express User Guide for the Memory-Mapped Interface with DMA**
- **Stratix V Hard IP for PCI Express User Guide for the Streaming Interface**

Configurations

The Arria 10 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 (DL)
- Transaction Layer (TL)

Optimized for Altera devices, the Arria 10 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. When configured as an Endpoint, the Arria 10 Hard IP for PCI Express using the Avalon-MM supports memory read and write requests and completions with or without data.

Figure 1-1: PCI Express Application with a Single Root Port and Endpoint

The following figure shows a PCI Express link between two Arria 10 FPGAs. One is configured as a Root Port and the other as an Endpoint.

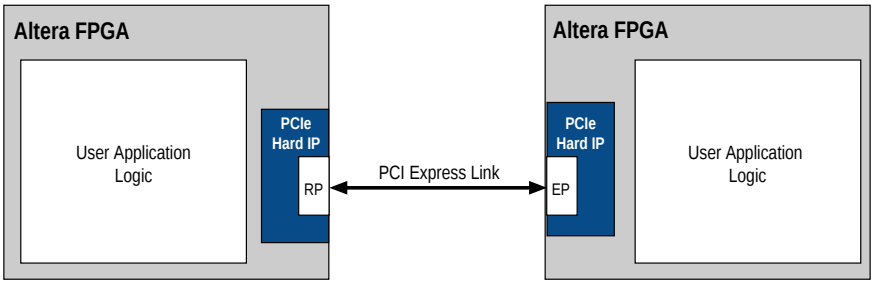
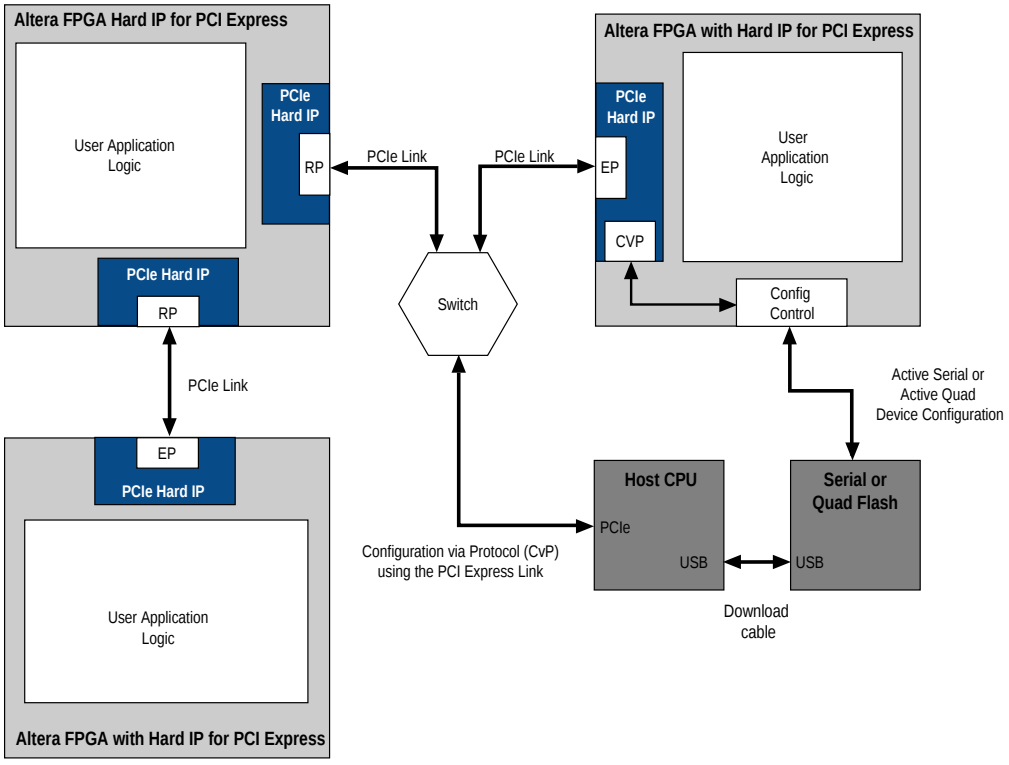


Figure 1-2:

The following figure illustrates a Arria 10 design that includes the following components:

- A Root Port that connects directly to a second FPGA that includes an Endpoint.
- Two Endpoints that connect to a PCIe switch.
- A host CPU that implements CvP using the PCI Express link connects through the switch. For more information about configuration over a PCI Express link, refer to "Configuration via Protocol (CvP)" on page 12–1.



Debug Features

The Arria 10 Hard IP for PCI Express includes debug features that allow observation and control of the Hard IP for faster debugging of system-level problems.

IP Core Verification

To ensure compliance with the PCI Express specification, Altera performs extensive validation of the Arria 10 Hard IP Core for PCI Express. The simulation environment uses multiple testbenches that consist of industry-standard bus functional models (BFMs) driving the PCI Express link 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

Compatibility Testing Environment

Altera has performed significant hardware testing of the Arria 10 Hard IP for PCI Express to ensure a reliable solution. In addition, Altera internally tests every release with motherboards and PCI Express switches from a variety of manufacturers. All PCI-SIG compliance tests are also run with each IP core release.

Performance and Resource Utilization

Because the IP core is implemented in hardened logic, it uses less than 1% of Arria 10 resources.

Both Avalon-MM variants include a bridge implemented in soft logic that functions as a front end to the Arria 10 Hard IP for PCI Express IP core. The following table shows the typical expected device resource utilization for selected configurations of the Arria 10 Hard IP for PCI Express using the current version of the Quartus II software targeting a Arria 10 device. With the exception of M20K memory blocks, the numbers of ALMs and logic registers in the following tables are rounded up to the nearest 50. Resource utilization numbers reflect changes to the resource utilization reporting starting in the Quartus II software v12.1 release 28 nm device families and upcoming device families.

Table 1-5: Performance and Resource Utilization Avalon-MM Hard IP for PCI Express

Data Rate or Interface Width	ALMs	Memory M20K	Logic Registers
Avalon-MM Bridge			
Gen1 ×4	1100	17	1500
Gen2 ×8	1900	25	2900
Avalon-MM Interface– Gen2 and Gen3 x8 256-Bit DMA			

Data Rate or Interface Width	ALMs	Memory M20K	Logic Registers
64	1100	14	1650
128	1750	19	2600
Avalon-MM Interface–Burst Capable Completer Only			
64	650	8	1000
128	1400	12	2400
Avalon-MM–Completer Only Single DWord			
64	250	0	350

Note: Soft calibration of the transceiver module requires additional logic. The amount of logic required depends upon the configuration.

Related Information

Fitter Resources Reports

Recommended Speed Grades

The following tables list the recommended speed grades for the supported interface widths, link widths, and Application Layer clock frequencies. When the Application Layer clock frequency is 250 MHz, Altera recommends setting the Quartus II Analysis & Synthesis Settings **Optimization Technique** to **Speed**.

For information about optimizing synthesis, refer to "*Setting Up and Running Analysis and Synthesis* in Quartus II Help. For more information about how to effect the **Optimization Technique** settings, refer to *Area and Timing Optimization* in volume 2 of the *Quartus II Handbook*.

Note: The ×2 variants are not available for the Arria 10 Hard IP for PCI Express in the current release.

Table 1-6: Arria 10 Recommended Speed Grades for All Avalon-MM Widths and Frequencies

Lane Rate	Link Width	Interface Width	Application Clock Frequency (MHz)	Recommended Speed Grades
Gen1	×1	64 bits	125	–1, –2, –3, –4
	×2	64 bits	125	–1, –2, –3, –4
	×4	64 bits	125	–1, –2, –3, –4
	×8	64 bits	250	–1, –2, –3 (2)
	×8	128 Bits	125	–1, –2, –3, –4

Lane Rate	Link Width	Interface Width	Application Clock Frequency (MHz)	Recommended Speed Grades
Gen2	×1	64 bits	62.5, 125	–1, –2, –3, –4
	×2	64 bits	125	–1, –2, –3 (2)
	×4	128 bits	125	–1, –2, –3, –4
	×8	128 bits	250	–1, –2, –3 (2)
Gen3	×1	64 bits	125	–1, –2, –3, –4
	×2	64 bits	250	–1, –2, –3 (2)
	×2	128 bits	125	–1, –2, –3 (2)
	×4	128 bits	250	–1, –2, –3 (2)
	×8	256 bits	250	–1, –2, –3 (2)

Notes:

1. This is a power-saving mode of operation.
2. The -4 speed grade is also possible for this configuration; however, it requires significant effort by the end user to close timing.

Related Information

- **Area and Timing Optimization**
- **Altera Software Installation and Licensing Manual**

Getting Started with the Avalon-MM Arria 10 Hard IP for PCI Express

2

December 2013

UG-01145_avmm

Subscribe Send Feedback

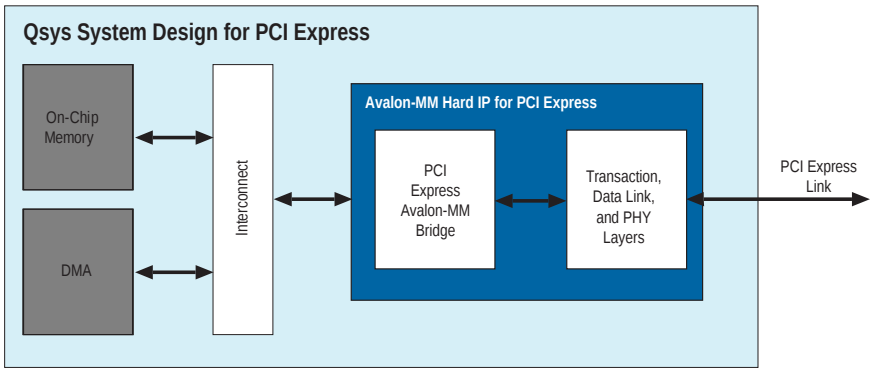
The design examples contain the following components:

- Avalon-MM Arria 10 Hard IP for PCI Express ×4 IP core
- On-Chip memory
- DMA controller

In the Qsys design flow you select the Avalon-MM Arria 10 Hard IP for PCI Express as a component. This component supports PCI Express Endpoint applications with bridging logic to convert PCI Express packets to Avalon-MM transactions and vice versa. The design example included in this chapter illustrates the use of an Endpoint with an embedded transceiver.

The following figure provides a high-level block diagram of the design example included in this release.

Figure 2-1: Qsys Generated Endpoint



As the figure illustrates, the design example transfers data between an on-chip memory buffer located on the Avalon-MM side and a PCI Express memory buffer located on the root complex side. The data transfer uses the DMA component which is programmed by the PCI Express software application running on the Root Complex processor.

Note: If you are already familiar with Qsys you can copy this example design from the installation directory and then begin with the *Simulating the Example Design* section below. This Qsys design example is available in the***<install_dir>*/ ip/altera/altera_pcie/altera_pcie_hip_ast_ed/example_design/a10** directory.

Related Information

Simulating the Example Design on page 2-11

Running Qsys

Follow these steps to launch Qsys:

1. Choose **Programs > Altera > Quartus II>** < *version_number* > (Windows Start menu) to run the Quartus II software. Alternatively, you can also use the Quartus II Web Edition software.
2. On the Quartus II File menu, click **New.**
3. Select **Qsys System File** and click **OK**. Qsys appears.
4. To establish global settings, click the **Project Settings** tab.
5. Specify the settings in the following table.

Table 2-1: Project Settings

Parameter	Value
Device family	**Arria 10**
Device	**10AX115S1F45I2SGES**
Clock crossing adapter type	**Handshake**
Limit interconnect pipeline stages to	**2**
Generation Id	**0**

Refer to *Creating a System with Qsys* in volume 1 of the *Quartus II Handbook* for more information about how to use Qsys, including information about the Project Settings tab. For an explanation of each Qsys menu item, refer to *About Qsys* in Quartus II Help.

Note: This example design requires that you specify the same name for the Qsys system as for the top-level project file. However, this naming is not required for your own design. If you want to choose a different name for the system file, you must create a wrapper HDL file that matches the project top level name and instantiate the generated system.

To add modules from the **Component Library** tab, under **Interface Protocols** in the **PCI** folder, click the **Avalon-MM Arria 10 Hard IP for PCI Express** component, then click **+Add**.

Related Information

- Creating a System with Qsys
- About Qsys

Customizing the Hard IP for PCI Express IP Core

The parameter editor uses bold headings to divide the parameters into separate sections. You can use the scroll bar on the right to view parameters that are not initially visible. Follow these steps to parameterize the Hard IP for PCI Express IP core:

1. Under the **System Settings** heading, specify the settings in the following table.

Table 2-2: System Settings

Parameter	Value
Lane rate	**Gen1 (2.5 Gbps)**
Number of lanes	**×4**
Port type	**Native endpoint**
Interface type	**Avalon-MM**
RX buffer credit allocation – performance for received requests	**Low**
Enable byte parity ports on Avalon-ST interface	**Off**
Enable multiple packets per cycle	**Off**
Enable configuration via PCI Express (CvP)	**Off**
Enable credit consumed selection port	**Off**
Enable Hard IP Reconfiguration	**Off**

2. Under the**Interface System Settings** heading, specify the settings in the following table:

Table 2-3: Interface System Settings

Parameter	Value
Application Interface	**64-bit**
Avalon-MM address width	**32-bit**
Peripheral mode	**Requestor/Completer**
Single DW Completer	**Off**
Control register access (CRA) Avalon-MM slave port	**On**
Enable multiple MSI/MSI-X support	**Off**
Auto enable PCIe interrupt (enabled at power-up)	**Off**
Number of address pages	**2**

Parameter	Value
Size of address pages	**1 MByte - 20 bits**

3. Under the **BAR Address Register** heading, specify the settings in the following table:

Table 2-4: PCI Base Address Registers (Type 0 Configuration Space)

BAR	BAR Type	BAR Size
0	**64-bit Prefetchable Memory**	**22**
1	**Disabled**	**0**
2	**32 bit Non-Prefetchable**	**15**
3–5	**Disabled**	**0**

4. For the **Device Identification Registers**, specify the values listed in the center column of the following table. The right-hand column of this table lists the value assigned to Altera devices. You must use the Altera values to run the Altera testbench. Be sure to use your company's values for your final product.

Table 2-5: Device Identification Registers

Parameter	Value	Altera Value
Vendor ID	`0x00000000`	0x00001172
Device ID	`0x00000000`	0x0000E001
Revision ID	`0x00000000`	0x00000001
Class Code	`0x00000000`	0x00FF0000
Subsystem Vendor ID	`0x00000000`	0x00001172
Subsystem Device ID	`0x00000000`	0x0000E001

5. Under the PCI Express and PCI Capabilities heading, specify the settings in the following table:

Table 2-6: PCI Express and PCI Capabilities

Parameter	Value
Device	
Maximum payload size	**128 Bytes**
Completion timeout range	**ABCD**
Implement completion timeout disable	Turn on this option

Parameter	Value
Error Reporting	
Advanced error reporting (AER)	Turn off this option
ECRC checking	Turn off this option
ECRC generation	Turn off this option
ECRC forwarding	Turn off this option
Track RX completion buffer overflow	Turn off this option
Link	
Link port number	**1**
Data Link Layer active reporting	Turn off this option
Surprise down reporting	Turn off this option
Slot clock configuration	Turn off this option
MSI	
Number of MSI messages requested	**4**
MSI-X	
Implement MSI X	Turn this option off
Slot	
Use slot register	Turn this option off
Slot power scale:	`0`
Slot power limit:	`0`
Slot number:	`0`
Power Management	
Endpoint L0s acceptable latency	**Maximum of 64 ns**
Endpoint L1 acceptable latency	**Maximum of 1 us**

Table 2-7: PHY Characteristics

Parameter	Value
Gen2 transmit deemphasis	**6dB**

6. Under the PHY Characteristics heading specify the settings in the following table:

Parameter	Value
Gen2 TX de-emphasis	**6 dB**

7. Click **Finish**.
8. To rename the **Arria 10 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`.

Related Information

- **PCI Express-to-Avalon-MM Address Translation for 32-Bit Bridge** on page 5-11
- **Minimizing BAR Sizes and the PCIe Address Space** on page 5-12
- **Avalon-MM-to-PCI Express Address Translation Algorithm for 32-Bit Addressing** on page 5-14

Adding the Remaining Components to the Qsys System

This section describes adding the DMA controller and on-chip memory to your system.

1. On the **Component Library** tab, type the following text string in the search box:
   `DMA`
   Qsys filters the component library and shows all components matching the text string you entered.
2. Click **DMA Controller** and then click **+Add**. This component contains read and write master ports and a control port slave.
3. In the **DMA Controller** parameter editor, specify the parameters and conditions listed in the following table.

Table 2-8: DMA Controller Parameters

Parameter	Value
Width of the DMA length register	13
Enable burst transfers	Turn on this option
Maximum burst size	Select **128**
Data transfer FIFO depth	Select **32**
Construct FIFO from registers	Turn off this option
Advanced	
Allowed Transactions	Turn on all options

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

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

5. Click **On-Chip Memory (RAM or ROM)** and then click **+Add**. Specify the parameters listed in the following table.

Table 2-9: On-Chip Memory Parameters

Parameter	Value
Memory Type	
Type	Select **RAM (Writeable)**
Dual port access	Turn off this option
Single clock option	Not applicable
Read During Write Mode	Not applicable
Block type	**Auto**
Size	
Data width	**64**
Total memory size	**4096 Bytes**
Minimize memory block usage (may impact f_{MAX})	Not applicable
Read Latency	
Slave s1 latency	**1**
Slave s2 latency	Not applicable
ECC Parameters	
ECC	**Disabled**
Memory Initialization	
Initialize memory content	Turn off this option
Enable non-default initialization file	Not applicable
Enable In System Memory Content Editor feature	Turn off this option
Instance ID	Not required

Completing the Connections in Qsys

In Qsys, hovering the mouse over the **Connections** column displays the potential connection points between components, represented as dots on connecting wires. A filled dot shows that a connection is made; an open

dot shows a potential connection point. Clicking a dot toggles the connection status. If you make a mistake, you can select **Undo** from the Edit menu or type `Ctrl-z`.

By default, Qsys filters some interface types to simplify the image shown on the **System Contents** tab. Complete these steps to display all interface types:

1. Click the **Filter** tool bar button.
2. In the Filter list, select **All interfaces**.
3. Close the **Filters** dialog box.

To complete the design, create the following connections:

1. Connect the `Rxm_bar0` Avalon Memory-Mapped Master port to the onchip_memory2_0 `s1` Avalon Memory-Mapped slave port using the following procedure:
   a. Click the `Rxm_BAR0` port, then hover in the **Connections** column to display possible connections.
   b. Click the open dot at the intersection of the `onchip_mem2_0` `s1` port and the pci_express_compiler `Rxm_bar0` to create a connection.
2. Repeat this procedure to make the connections listed in the following table.

Table 2-10: Qsys Connections

Make Connection From:	To:
DUT `app_nreset_status` Reset Output	onchip_memory `reset1` Avalon slave port
DUT `app_nreset_status` Reset Output	dma_0 `reset` Reset Input
DUT `Rxm_bar0` Avalon Memory Mapped Master	onchip_memory `s1` Avalon slave port
DUT `Rxm_bar2` Avalon Memory Mapped Master	DUT `Cra` Avalon Memory Mapped Slave
DUT `Rxm_bar2` Avalon Memory Mapped Master	dma_0 `control_port_slave` Avalon Memory Mapped Slave
DUT `RxmIrq` Interrupt Receiver	dma_0 `irq` Interrupt Sender
DUT `Txs` Avalon Memory Mapped Slave	dma_0 `read_master` Avalon Memory Mapped Master
DUT `Txs` Avalon Memory Mapped Slave	dma_0 `write_master` Avalon Memory Mapped Master
onchip_memory `s1` Avalon Memory Mapped Slave	dma_0 `read_master` Avalon Memory Mapped Master
onchip_memory `s1` Avalon Memory Mapped Slave	dma_0 `write_master` Avalon Memory Mapped Master
DUT `app_nreset_status`	onchip_memory `reset1`
DUT `app_nreset_status`	dma_0 `reset`

Make Connection From:	To:
DUT nreset_status	clk0 clk_reset

Specifying Clocks and Interrupts

Complete the following steps to connect the clocks and specify interrupts:

1. To connect DUT coreclkout to the **onchip_memory** and **dma_0** clock inputs, click in the **Clock** column next to the DUT coreclkout clock input. Click **onchip_memory.clk1** and **dma_0.clk**.
2. To specify the interrupt number for DMA interrupt sender, irq, type 0 in the **IRQ** column next to the irq port.
3. On the File menu, click **Save.**

Specifying Exported Interfaces

Many interface signals in this Qsys system connect to modules outside the design. Follow these steps to export an interface:

1. Click in the **Export** column.
2. Accept the default name that appears in the **Export** column.

Table 2-11: Exported Interfaces

Interface Name	Exported Name
DUT refclk	pcie_a10_hip_0_refclk
DUT npor	pcie_a10_hip_0_npor
DUT hip_serial	pcie_a10_hip_0_hip_serial
DUT hip_pipe	pcie_a10_hip_0_hip_pipe
DUT hip_ctrl	pcie_a10_hip_0_hip_ctrl

Specifying Address Assignments

Qsys requires that you resolve the base addresses of all Avalon-MM slave interfaces in the Qsys system. You can either use the auto-assign feature, or specify the base addresses manually. To use the auto-assign feature, on the **System** menu, click **Assign Base Addresses**. In the design example, you assign the base addresses manually.

The Avalon-MM Arria 10 Hard IP for PCI Express assigns base addresses to each BAR.

Follow these steps to assign a base address to an Avalon-MM slave interface manually:

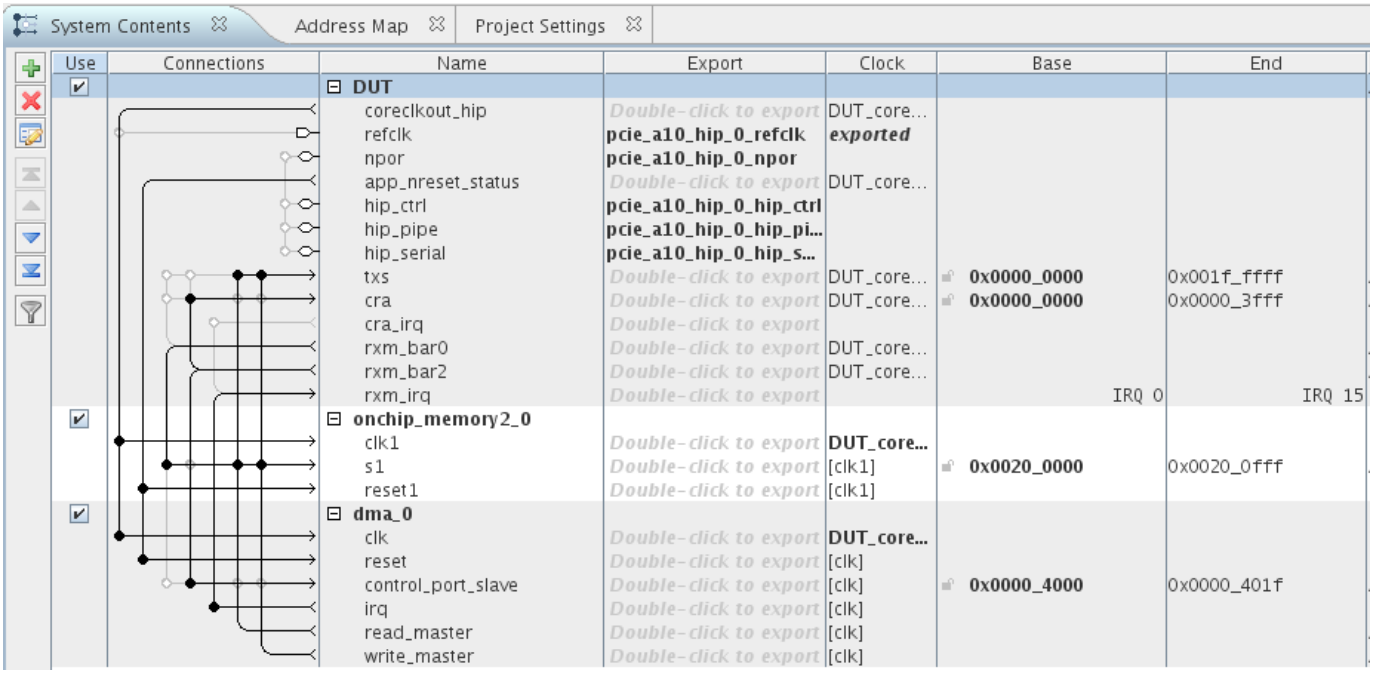
1. In the row for the Avalon-MM slave interface base address you want to specify, click the **Base** column.
2. Type your preferred base address for the interface.
3. Assign the base addresses listed in the following table.

Table 2-12: Base Address Assignments for Avalon-MM Slave Interfaces

Interface Name	Exported Name
DUT `Txs`	`0x00000000`
DUT `Cra`	`0x00000000`
`DMA control_port_slave`	`0x00004000`
`onchip_memory_0 s1`	`0x00200000`

The following figure illustrates the complete system.

Figure 2-2: Complete PCI Express Example Design



For this example BAR1:0 is 22 bits or 4 MB. This BAR accesses Avalon addresses from 0x00200000–0x00200FFF. BAR2 is 15 bits or 32 KBytes. BAR2 accesses the DMA control_port_slave at offsets 0x00004000 through 0x0000403F. The pci_express CRA slave port is accessible at offsets 0x0000000–0x0003FFF from the programmed BAR2 base address.

Related Information

Minimizing BAR Sizes and the PCIe Address Space on page 5-12

Simulating the Example Design

Follow these steps to generate the files for the testbench and synthesis.

1. Click the **Generate** menu. The **Generate**dialog box appears.
2. Under **Simulation** section, set the following options:
   a. For **Create simulation model**, select **None**. (This option allows you to create a simulation model for inclusion in your own custom testbench.)
   b. For **Create testbench Qsys system**, select **Standard, BFMs for standard Qsys interfaces**.
   c. For **Create testbench simulation model**, select **Verilog**.
3. For HDL design files for synthesis: select: **Verilog**
4. Turn on **Create block symbol file (.bsf)**.
5. Click **Generate**.
6. After Qsys reports **Generate Completed** in the **Generate** progress box title, click **Close**.
7. On the **File** menu, click **Save**. and type the file name `ep_g1x4.qsys`.

The following table lists the directories that are generated in your Quartus II project directory.

Table 2-13: Qsys System Generated Directories

Directory	Location
Qsys system	**<*project_dir*>/ep_g1x4_avmm64**
Testbench	**<*project_dir*>/ep_g1x4_avmm64/testbench**
Synthesis	**<*project_dir*>/ep_g1x4_avmm64/synthesis**

Qsys creates a top-level testbench named <*project_dir*>**/ep_g1x4_avmm64/testbench/ep_g1x4_tb.qsys**. This testbench connects an appropriate BFM to each exported interface. Qsys generates the required files and models to simulate your PCI Express system.

The simulation of the design example uses the following components and software:

- The system you created using Qsys
- A testbench. You can view this testbench in Qsys by opening <*project_dir*>**/ep_g1_x4_avmm64/testbench/**<*dev*>**_avmm_tb.qsys/**.
- The ModelSim software

Note: You can also use any other supported third-party simulator to simulate your design.

Qsys creates IP functional simulation models for all the system components. The IP functional simulation models are the **.vo** or **.vho** files generated by Qsys in your project directory.

For more information about IP functional simulation models, refer to *Simulating Altera* Designs in volume 3 of the *Quartus II Handbook*.

Complete the following steps to run the Qsys testbench:

1. In a terminal window, change to the <*project_dir*>/**ep_g1x4_avmm64/testbench/mentor** directory.
2. Start the ModelSim simulator.
3. To run the simulation, type the following commands in a terminal window:

   a. `do msim_setup.tcl`
   b. `ld_debug`
   c. `run 140000 ns`

The driver performs the following transactions with status of the transactions displayed in the ModelSim simulation message window:

1. Various configuration accesses to the Avalon-MM Arria 10 Hard IP for PCI Express in your system after the link is initialized
2. Setup of the Address Translation Table for requests that are coming from the DMA component
3. Setup of the DMA controller to read 512 Bytes of data from the Transaction Layer Direct BFM shared memory
4. Setup of the DMA controller to write the same data back to the Transaction Layer Direct BFM shared memory
5. Data comparison and report of any mismatch

The following example shows the transcript from a successful simulation run.

Example 2-1: Transcript from ModelSim Simulation of Gen1 x4 Endpoint

```
# INFO:  3657 ns  RP LTSSM State: DETECT.ACTIVE
# INFO:  4425 ns  RP LTSSM State: POLLING.ACTIVE
# INFO: 17257 ns  RP LTSSM State: DETECT.QUIET
# INFO: 17353 ns  RP LTSSM State: DETECT.ACTIVE
# INFO: 17405 ns  RP LTSSM State: DETECT.QUIET
# INFO: 17485 ns  RP LTSSM State: DETECT.ACTIVE
# INFO: 18249 ns  RP LTSSM State: POLLING.ACTIVE
# INFO: 31081 ns  RP LTSSM State: DETECT.QUIET
# INFO: 31177 ns  RP LTSSM State: DETECT.ACTIVE
# INFO: 31229 ns  RP LTSSM State: DETECT.QUIET
# INFO: 31357 ns  RP LTSSM State: DETECT.ACTIVE
# INFO: 32073 ns  RP LTSSM State: POLLING.ACTIVE
# INFO: 38677 ns  EP LTSSM State: DETECT.ACTIVE
# INFO: 44905 ns  RP LTSSM State: DETECT.QUIET
# INFO: 45813 ns  EP LTSSM State: DETECT.QUIET
# INFO: 46073 ns  RP LTSSM State: DETECT.ACTIVE
# INFO: 46793 ns  RP LTSSM State: POLLING.ACTIVE
# INFO: 47093 ns  EP LTSSM State: DETECT.ACTIVE
# INFO: 54213 ns  EP LTSSM State: POLLING.ACTIVE
# INFO: 56517 ns  EP LTSSM State: POLLING.CONFIG
# INFO: 59625 ns  RP LTSSM State: DETECT.QUIET
# INFO: 62841 ns  RP LTSSM State: DETECT.ACTIVE
# INFO: 63561 ns  RP LTSSM State: POLLING.ACTIVE
# INFO: 76393 ns  RP LTSSM State: POLLING.CONFIG
```

```
# INFO: 77545 ns  RP LTSSM State: CONFIG.LINKWIDTH.START
# INFO: 77829 ns  EP LTSSM State: CONFIG.LINKWIDTH.START
# INFO: 78469 ns  EP LTSSM State: CONFIG.LINKWIDTH.ACCEPT
# INFO: 78905 ns  RP LTSSM State: CONFIG.LINKWIDTH.ACCEPT
# INFO: 79225 ns  RP LTSSM State: CONFIG.LANENUM.WAIT
# INFO: 80377 ns  RP LTSSM State: CONFIG.LANENUM.ACCEPT
# INFO: 80421 ns  EP LTSSM State: CONFIG.LANENUM.ACCEPT
# INFO: 80697 ns  RP LTSSM State: CONFIG.LANENUM.WAIT
# INFO: 80889 ns  RP LTSSM State: CONFIG.LANENUM.ACCEPT
# INFO: 81209 ns  RP LTSSM State: CONFIG.COMPLETE
# INFO: 81701 ns  EP LTSSM State: CONFIG.COMPLETE
# INFO: 83017 ns  RP LTSSM State: CONFIG.IDLE
# INFO: 83109 ns  EP LTSSM State: CONFIG.IDLE
# INFO: 83273 ns  RP LTSSM State: L0
# INFO: 83429 ns  EP LTSSM State: L0
  . . .
# INFO: 106240 ns Completed configuration of Endpoint BARs.
# INFO: 107792 ns Starting Target Write/Read Test.
# INFO: 107792 ns Target BAR = 0
# INFO: 107792 ns Length = 000512,Start Offset=000000
# INFO: 111896 ns Target Write and Read compared okay
# INFO: 111896 ns Starting DMA Read/Write Test.
# INFO: 111896 ns Setup BAR = 2
# INFO: 111896 ns Length = 000512, Start Offset = 000000
# INFO: 117148 ns Interrupt Monitor: Interrupt INTA Asserted
# INFO: 117148 ns Clear Interrupt INTA
# INFO: 118380 ns Interrupt Monitor: Interrupt INTA Deasserted
# INFO: 124860 ns MSI recieved!
# INFO: 124860 ns   DMA Read and Write compared okay!
# SUCCESS: Simulation stopped due to successful completion!
# Break in Function ebfm_log_stop_sim at
./..//ep_g1x4_avmm64_tb/simulation/submodules//altpcietb_bfm_log.v
 line 78
```

Related Information

Simulating Altera Designs

Simulating the Single DWord Design

You can use the same testbench to simulate the **Completer-Only single dword** IP core by changing the settings in the driver file. Complete the following steps for the Verilog HDL design example:

1. In a terminal window, change to the *<project_dir>*/*<variant>*/**testbench**/*<variant>*_**tb/simulation/submodules** directory.
2. Open **altpcietb_bfm_driver_avmm.v** file your text editor.
3. To enable target memory tests and specify the completer-only single dword variant, specify the following parameters:

a. `parameter RUN_TGT_MEM_TST = 1;`
b. `parameter RUN_DMA_MEM_TST = 0;`
c. `parameter AVALON_MM_LITE = 1;`

4. Change to the < *project_dir* >/<*variant*> / **testbench /mentor** directory.
5. Start the ModelSim simulator.
6. To run the simulation, type the following commands in a terminal window:

a. `do msim_setup.tcl`
b. `ld_debug` (The -debug suffix stops optimizations, improving visibility in the ModelSim waveforms.)
c. `run 140000 ns`

Understanding Channel Placement Guidelines

Arria 10 transceivers are organized in banks of six channels. The transceiver bank boundaries are important for clocking resources, bonding channels, and fitting. Refer to the *Physical Placement of the Arria 10 Hard IP for PCI Express IP and Channels* for illustrations of channel placement for ×1, ×4, and ×8 variants.

Related Information

Physical Placement of the Arria 10 Hard IP for PCIe IP and Channels on page 4-1

Adding Synopsis Design Constraints

Before you can compile your design using the Quartus II software, you must add a few Synopsys Design Constraints (SDC) to your project. Complete the following steps to add these constraints:

1. Browse to <*project_dir*>**/ep_g1x4_avmm64/synthesis/submodules**
2. Add the constraints shown in the following Example **altera_pci_express.sdc**.

Note: Because **altera_pci_express.sdc** is overwritten each time you regenerate your design, you should save a copy of this file in an additional directory that the Quartus II software does not overwrite.

Example 2-2: Synopsys Design Constraints

```
create_clock -period “100 MHz” -name {refclk_pci_express}
{*refclk_*}
derive_pll_clocks
derive_clock_uncertainty
```

Creating a Quartus II Project

You can create a new Quartus II project with the New Project Wizard, which helps you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity. To create a new project follow these steps:

1. On the Quartus II File menu, click **New,** then **New Quartus II Project**, then **OK**.
2. Click **Next** in the **New Project Wizard: Introduction** (The introduction does not appear if you previously turned it off.)
3. On the **Directory, Name, Top-Level Entity** page, enter the following information:
   a. For What is the working directory for this project, browse to < *project_dir* > **/ep_g1x4_avmm64/synthesis/**
   b. For **What is the name of this project**, select **ep_g1x4_avmm64** from the **synthesis** directory
4. Click **Next**.
5. On the **Add Files** page, add < *project_dir* > **/ep_g1x4/_avmm64/synthesis/ep_g1_x4.qip** to your Quartus II project. This file lists all necessary files for Quartus II compilation, including the **altera_pci_express.sdc** that you just modified.
6. Click **Next** to display the **Family & Device Settings** page.
7. On the **Device** page, choose the following target device family and options:
   a. In the **Family** list, select **Arria 10**.
   b. In the **Devices** list, select **All**.
   c. In the **Available devices** list, select **10AX115S1F45I2SGES**.
8. Click **Next** to close this page and display the **EDA Tool Settings** page.
9. From the **Simulation** list, select **ModelSim®**. From the **Format** list, select the HDL language you intend to use for simulation.
10. Click **Next** to display the **Summary** page.
11. Check the **Summary** page to ensure that you have entered all the information correctly.

Compiling the Design

Follow these steps to compile your design:

1. On the Quartus II Processing menu, click **Start Compilation**.
2. After compilation, expand the **TimeQuest Timing Analyzer** folder in the Compilation Report. Note whether the timing constraints are achieved in the Compilation Report.

If your design does not initially meet the timing constraints, you can find the optimal Fitter settings for your design by using the Design Space Explorer. To use the Design Space Explorer, click **Launch Design Space Explorer** on the tools menu.

Programming a Device

After you compile your design, you can program your targeted Altera device and verify your design in hardware.

For more information about programming Altera FPGAs, refer to *Quartus II* Programmer.

Related Information

Quartus II Programmer

Parameter Settings 3

December 2013

UG-01145_avmm

System Settings

The first group of settings defines the overall system.

Table 3-1: System Settings for PCI Express

Parameter	Value	Description
Lane Rate	**Gen1 (2.5 Gbps)** **Gen2 (2.5/5.0 Gbps)** **Gen3 (2.5/5.0/8.0 Gbps)**	Specifies the maximum data rate at which the link can operate.
Number of Lanes	**×1, ×2, ×4, ×8**	Specifies the maximum number of lanes supported.
Port type	**Native Endpoint** **Root Port** **Legacy Endpoint**	Specifies the port type. Altera recommends **Native Endpoint** for all new Endpoint designs. The Endpoint stores parameters in the Type 0 Configuration Space. The Root Port stores parameters in the Type 1 Configuration Space.
Interface type	**Avalon-ST** **Avalon-MM**	Selects either the Avalon-ST or Avalon-MM interface.
Application interface	**64-bit** **128-bit** **256-bit**	Specifies the interface between the PCI Express Transaction Layer and the Application Layer. Refer to **coreclkout_hip** on page 8-6 for a comprehensive list of available link width, interface width, and frequency combinations. For the Avalon-MM interface with DMA, only 256-bit interface is available.
RX Buffer credit allocation -	**Minimum** **Low**	Determines the allocation of posted header credits, posted data credits, non-posted header credits, completion header credits,

Parameter	Value	Description
performance for received requests	**Medium** **High** **Maximum**	and completion data credits in the 16 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 the *Flow Control* chapter for more information about optimizing performance. The Flow Control chapter explains how the **RX credit allocation** and the **Maximum payload RX Buffer 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 on the **Device** tab. The **Message** window of the GUI dynamically updates the number of credits for Posted, Non-Posted Headers and Data, and Completion Headers and Data as you change this selection. • **Minimum RX Buffer credit allocation -performance for received requests)**–This setting configures the minimum PCIe specification allowed for 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. • **Balanced**–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.

Parameter	Value	Description
		• **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.
Enable byte parity ports on Avalon ST interface	**On/Off**	When **On**, the RX and TX datapaths are parity protected. Parity is odd.
Enable multiple packets per cycle	**On/Off**	When **On**, the 256-bit Avalon-ST interface supports the transmission of TLPs starting at any 128-bit address boundary, allowing support for multiple packets in a single cycle. To support multiple packets per cycle, the Avalon-ST interface includes 2 start of packet and end of packet signals for the 256-bit Avalon-ST interfaces. This feature is only supported for Gen3 ×8. For more information refer to "Multiple Packets per Cycle" on page 8–27.
Enable configuration via Protocol (CvP)	**On/Off**	When **On**, the Quartus II software places the Endpoint in the location required for configuration via protocol (CvP). For more information about CvP, click the *Configuration via Protocol (CvP)* link below. CvP is not supported for Gen3 variants.
Enable credit consumed selection port	**On/Off**	When you turn on this option, the core includes the `tx_cons_cred_sel` port.
Enable Hard IP Reconfiguration	**On/Off**	When **On**, you can use the Hard IP reconfiguration bus to dynamically reconfigure Hard IP read-only registers. For more information refer to *Hard IP Reconfiguration Interface*.
Enable Hard IP Reconfiguration	**On/Off**	When **On**, you can use the Hard IP reconfiguration bus to dynamically reconfigure Hard IP read-only registers. For more information refer to *Hard IP Reconfiguration Interface*. This parameter is not available fo the Avalon-MM IP Cores.

Related Information

- **Type 0 Configuration Space Registers** on page 7-6
- **Type 1 Configuration Space Registers** on page 7-7

- **coreclkout_hip** on page 8-6
- **Throughput Optimization** on page 11-1
- **PCI Express Base Specification 3.0**
- **Configuration via Protocol (CvP)** on page 14-1

Base Address Register (BAR) and Expansion ROM Settings

The following table lists the PCI BAR and Expansion ROM register settings. As this table indicates, the type and size of BARs available depend on port type.

Table 3-2: BAR Registers

Parameter	Value	Description
Type	**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 memories. The **I/O address space** BAR is only available for the **Legacy Endpoint**.
Size	**16 Bytes–8 EBytes**	Supports the following memory sizes: • 128 bytes–2 GBytes or 8 EBytes: **Endpoint** and **Root Port** variants • 6 bytes–4 KBytes: **Legacy Endpoint** variants
Expansion ROM	**Disabled–16 MBytes**	Specifies the size of the optional ROM. The expansion ROM is not available for the Avalon-MM Arria 10 Hard IP for PCI Express.

Base and Limit Registers for Root Ports

The following table describes the `Base` and `Limit` registers which are available in the Type 1 Configuration Space for Root Ports. These registers are used for TLP routing and specify the address ranges assigned to components that are downstream of the Root Port or bridge.

Table 3-3: 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.

Related Information

PCI to PCI Bridge Architecture Specification

Device Identification Registers

The following table 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 optional reconfiguration block signals.

Table 3-4: Device ID Registers

Register Name	Range	Default Value	Description
Vendor ID	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. Address offset: 0x000.
Device ID	16 bits	0x00000000	Sets the read-only value of the `Device ID` register. Address offset: 0x000.
Revision ID	8 bits	0x00000000	Sets the read-only value of the `Revision ID` register. Address offset: 0x008.
Class code	24 bits	0x00000000	Sets the read-only value of the `Class Code` register. Address offset: 0x008.
Subsystem Vendor ID	16 bits	0x00000000	Sets the read-only value of the `Subsystem Vendor ID` register in the PCI Type 0 Configuration Space. This parameter cannot be set to 0xFFFF per the *PCI Express Base Specification*. Address offset: 0x02C.

Register Name	Range	Default Value	Description
Subsystem Device ID	16 bits	0x00000000	Sets the read-only value of the `Subsystem Device ID` register in the PCI Type 0 Configuration Space. Address offset: 0x02C

Related Information

PCI Express Base Specification 3.0

PCI Express and PCI Capabilities Parameters

This group of parameters defines various capability properties of the IP core. Some of these parameters are stored in the PCI Configuration Space - PCI Compatible Configuration Space. The byte offset within the PCI Configuration Space - PCI Compatible Configuration Space indicates the parameter address.

Device Capabilities

Table 3-5: Capabilities Registers

Parameter	Possible Values	Default Value	Description
Maximum payload size	**128 bytes** **256 bytes** **512 bytes** **1024 bytes** **2048 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]). Address: 0x084.

Parameter	Possible Values	Default Value	Description
Tags supported [1]	**32** **64**	32 - Avalon-ST	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 9–9 on page 9–5. The Transaction Layer tracks all outstanding completions for non-posted requests made by the Application Layer. This parameter configure**Tags supported**es 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 on the `tl_cfg_ctl` output signal 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* . For all other functions this field is reserved and must be hardwired to 0x0000b. Four time value ranges are defined: • Range A: 50 us 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. The function must implement a timeout value in the range 50 s to 50 ms. The following values are used to specify the range: • None – Completion timeout programming is not supported • 0001 Range A • 0010 Range B

(1) The Arria 10 Hard IP for PCI Express using the Avalon-MM interface always supports 8 tags. You do not need to configure this parameter.

Parameter	Possible Values	Default Value	Description
			• 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.
Disable completion timeout	**On/Off**	On	Disables the completion timeout mechanism. 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 3-6: Error Reporting

Parameter	Value	Default Value	Description
Advanced error reporting (AER)	**On/Off**	Off	When **On**, enables the Advanced Error Reporting (AER) capability.
ECRC check	**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. Not applicable for Avalon-MM DMA.
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. Not applicable for Avalon-MM DMA.

Parameter	Value	Default Value	Description
Track Receive Completion Buffer Overflow	**On/Off**	Off	When **On**, the core includes the `rxfx_cplbuf_ovf` output status signal to track the RX posted completion buffer overflow status. Not applicable for Avalon-MM DMA.

Note: :

1. Throughout this user guide, the terms word, dword and qword have the same meaning that they have in the *PCI Express Base Specification*. A word is 16 bits, a dword is 32 bits, and a qword is 64 bits.

Link Capabilities

Table 3-7: Link Capabilities

Parameter	Value	Description
Link port number	**0x01**	Sets the read-only value of the port number field in the `Link Capabilities Register`.
Data link layer active reporting	**On/Off**	Turn **On** this parameter 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 parameter 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 for the Arria 10 Hard IP for PCI Express in Root Port mode. Not applicable for Avalon-MM DMA.
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 for the Arria 10 Hard IP for PCI Express in Root Port mode. Not applicable for Avalon-MM DMA.
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.

MSI and MSI-X Capabilities

Table 3-8: MSI and MSI-X Capabilities

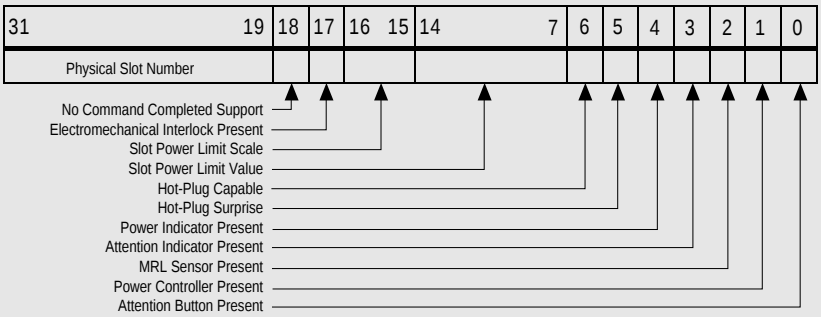
Parameter	Value	Description
MSI messages requested	**1, 2, 4, 8, 16, 32**	Specifies the number of messages the Application Layer can request. Sets the value of the `Multiple Message Capable` field of the `Message Control` register, 0x050[31:16].
	MSI-X Capabilities	
Implement MSI-X	**On/Off**	When **On**, enables the MSI-X functionality.
	Bit Range	
MSI-X Table size	[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}). Address offset: 0x068[26:16]
MSI-X Table Offset	[31:0]	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 (1). This field is read-only.
MSI-X Table BAR Indicator	[2:0]	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.
MSI-X Pending Bit Array (PBA) Offset	[31:0]	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.
MSI-X Pending Bit Array (PBA) Offset-BAR Indicator	[2:0]	Specifies the function Base Address registers, located beginning at 0x10 in Configuration Space, that maps the MSI-X PBA into memory space. This field is read-only. Legal range is 0–5.

Note:

1. Throughout this user guide, the terms word, dword and qword have the same meaning that they have in the *PCI Express Base Specification*. A word is 16 bits, a dword is 32 bits, and a qword is 64 bits.

Slot Capabilities

Table 3-9: Slot Capabilities

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 supported in Root Port mode. Not applicable for Avalon-MM DMA.
Slot capability register	—	Defines the characteristics of the slot. You turn on this option by selecting **Enable slot capability**. The various bits are defined as follows: 
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* for more information.
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* for more information.
Slot number	`0-8191`	Specifies the slot number.

Power Management

Table 3-10: Power Management Parameters

Parameter	Value	Description
Endpoint L0s acceptable latency	**Maximum of 64 ns** **Maximum of 128 n** **Maximum of 256 ns** **Maximum of 512 ns** **Maximum of 1 us** **Maximum of 2 us** **Maximum of 4 us** **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). This Endpoint 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	**Maximum of 1 us** **Maximum of 2 us** **Maximum of 4 us** **Maximum of 8 us** **Maximum of 16 us** **Maximum of 32 us** **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`. This Endpoint 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.

PHY Characteristics

Table 3-11: PHY Characteristics

Parameter	Value	Description
Gen2 transmit deemphasis	**3.5dB** **6dB**	Specifies the transmit deemphasis for Gen2. Altera recommends the following settings: • 3.5dB: Short PCB traces • 6.0dB: Long PCB traces.

Physical Placement of the Arria 10 Hard IP for PCIe IP and Channels

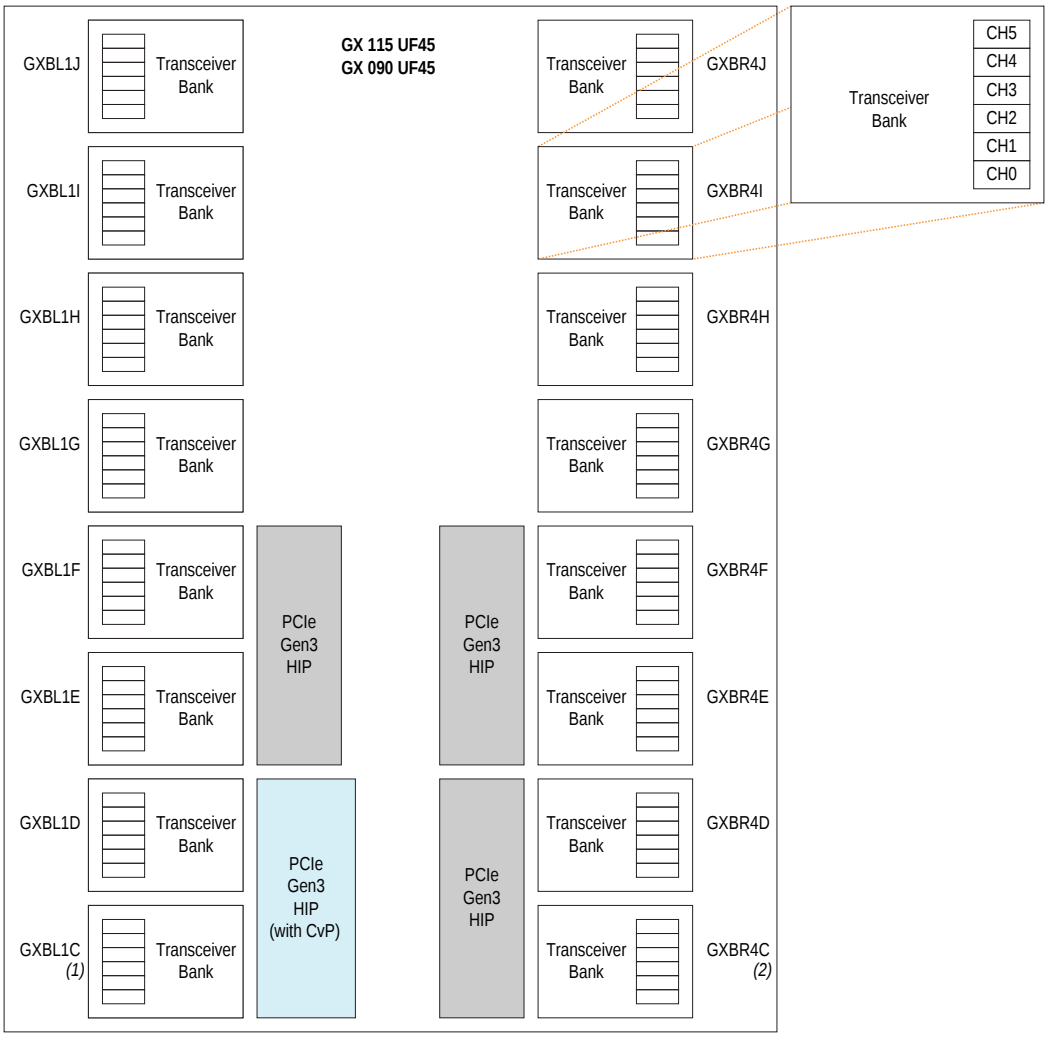
4

Subscribe Send Feedback

Physical Layout of Hard IP In Arria 10 Devices

Arria 10 devices include from 1–4 hard IP blocks for PCI Express. The following figures illustrate the placement of the hard IP for PCI Express and transceiver banks in Arria 10 devices. Note that the bottom left hard IP block includes the CvP functionality.

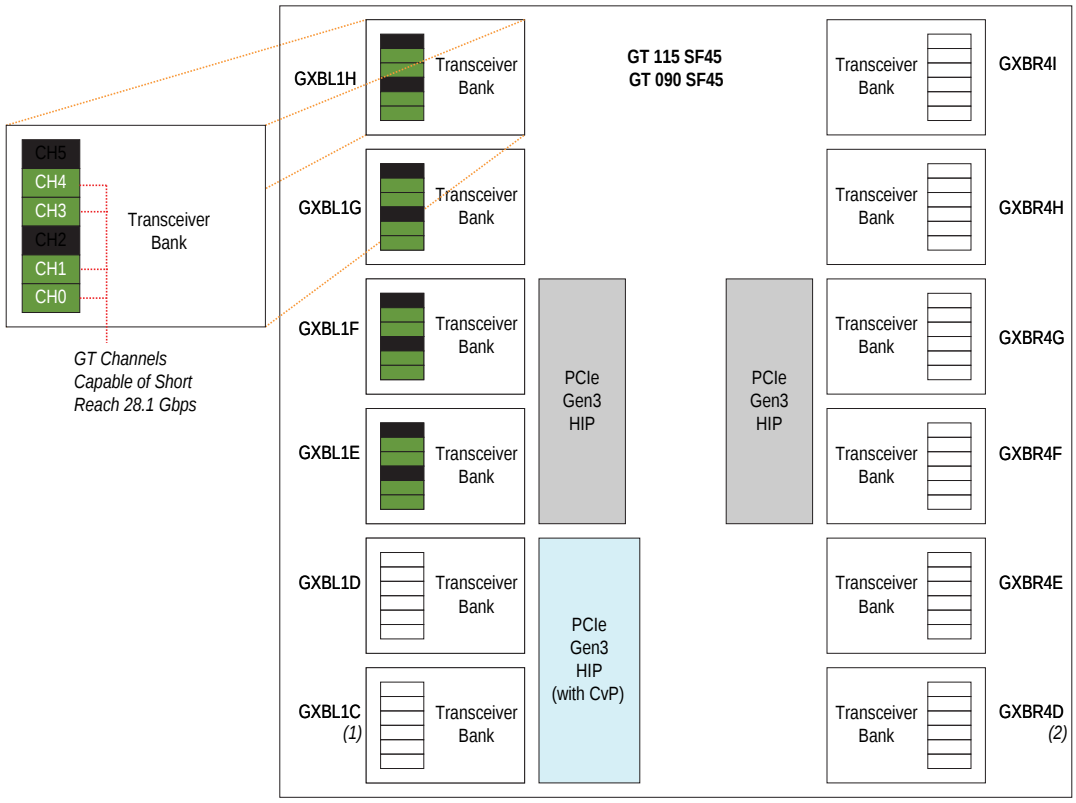
Figure 4-1: Arria 10 Devices with 96 Transceiver Channels and Four PCIe Hard IP Blocks



Notes:
(1) Nomenclature of left column bottom transceiver banks always begins with "C"
(2) Nomenclature of right column bottom transceiver banks may begin with "C", "D", or "E".

Legend:
- PCIe Gen3 HIP blocks with CvP capabilities
- PCIe Gen3 HIP blocks without CvP capabilities

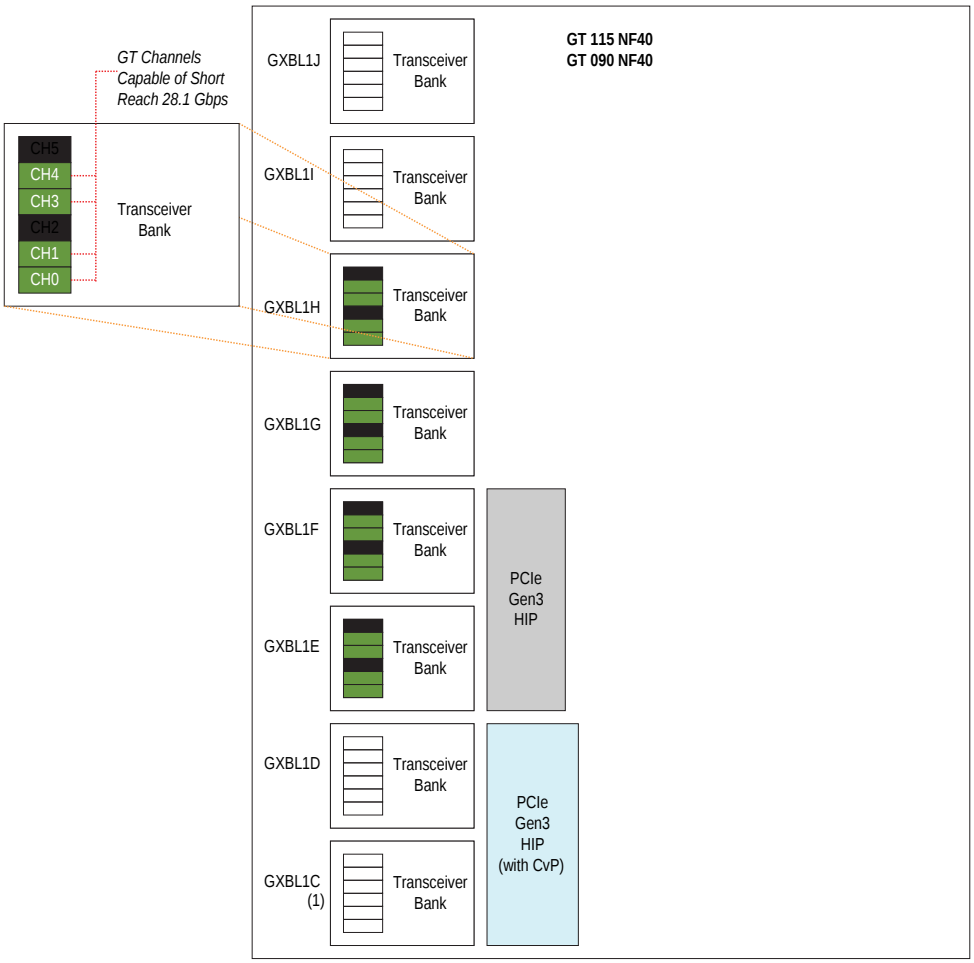
Figure 4-2: Arria 10 Devices with 72 Transceiver Channels and Three PCIe Hard IP Blocks



Notes:
(1) Nomenclature of left column bottom transceiver banks always begins with "C"
(2) Nomenclature of right column bottom transceiver banks may begin with "C", "D", or "E".

Legend:
- GT transceiver channels (channel 0, 1, 3, and 4)
- Transceiver channels that cannot be used (channel 2 and 5) when all four GT channels are used.
- PCIe Gen3 HIP blocks with CvP capabilities
- PCIe Gen3 HIP blocks without CvP capabilities

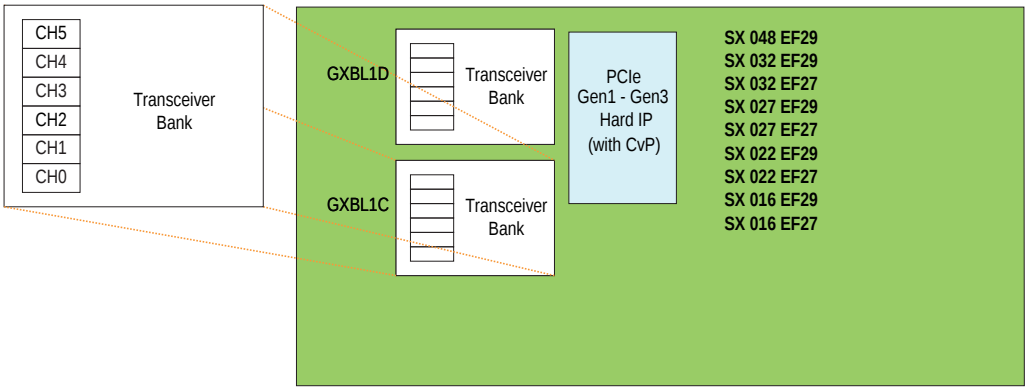
Figure 4-3: Arria 10 GT Devices with 48 Transceiver Channels and Two PCIe Hard IP Blocks



Notes:
(1) Nomenclature of left column bottom transceiver banks always begins with "C"
(2) These devices have transceivers only on left hand side of the device.

Legend:
- GT transceiver channels (channel 0, 1, 3, and 4)
- Transceiver channels that cannot be used (channel 2 and 5) when all four GT channels are used.
- PCIe Gen3 HIP blocks with CvP capabilities
- PCIe Gen3 HIP blocks without CvP capabilities

Figure 4-4: Arria 10 GT Devices with 6 or 12 Transceiver Channels and One PCIe Hard IP Block



Note:
(1) These devices have transceivers only on the left hand side of the device.

Legend:
- PCIe Gen 3 HIP blocks with CvP capabilities
- Arria 10 SX device with 12 transceiver channels and one Hard IP block

Refer to the *Arria 10 Transceiver Layout* in the *Arria 10 Transceiver PHY User Guide* for comprehensive figures for Arria 10 GT, GX, and SX devices.

Related Information

Arria 10 Hard IP for PCI Express User Guide

Channel Placement for PCIe In Arria 10 Devices

The following figures illustrate the x1, x2, x4, and x8 channel placement for the Arria 10 Hard IP for PCI Express. Gen1 and Gen2 variants use the ATX PLL as the TX PLL. In these figures, channels that are not used for the PCI Express protocol are available for other protocols. Unused channels are shown in gray.

Because Gen3 variants must initially train at the Gen1 data rate, Gen3 variants require 2 PLLs to generate the 2.5 Gbps and 8.0 Gbps clocks. Gen3 variants use a fractional PLL for the Gen1 data rate and an ATX PLL for the Gen3 data rate.

Note: You cannot change channel placement illustrated below.

Figure 4-5: Arria 10 Gen1 and Gen2 x1 Channel Placement

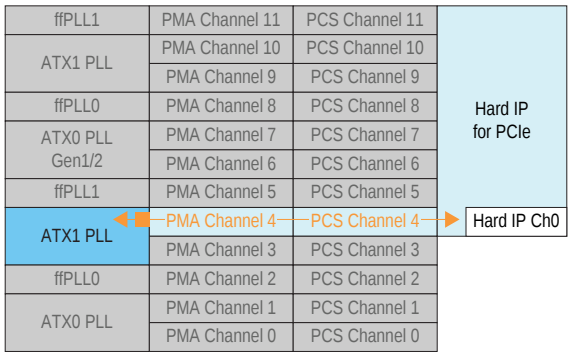


Figure 4-6: Arria 10 Gen1 and Gen2 x2 Channel Placement

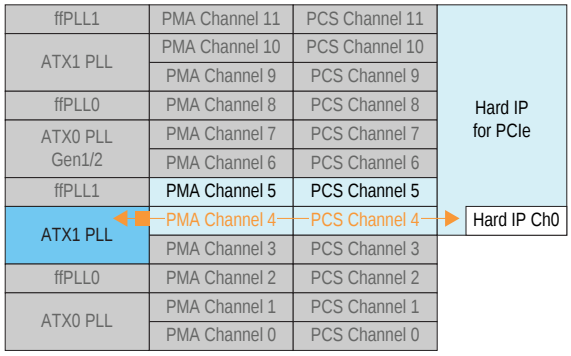


Figure 4-7: Arria 10 Gen1 and Gen2 x4 Channel Placement

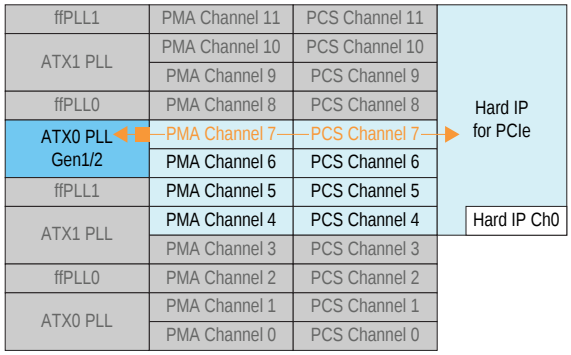


Figure 4-8: Gen1 and Gen2 x8 Channel Placement

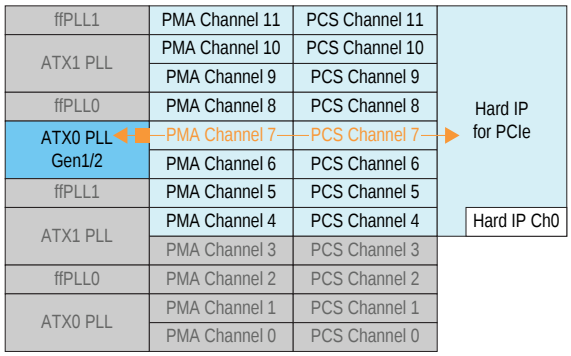


Figure 4-9: Arria 10 Gen3 x1 Channel Placement

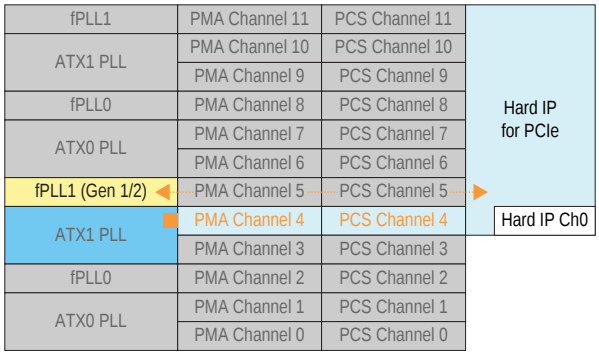


Figure 4-10: Arria 10 Gen3 x2 Channel Placement

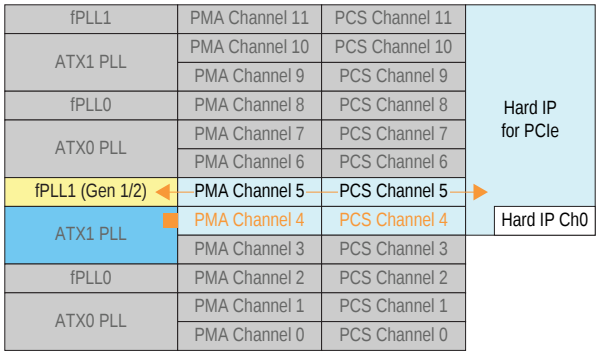


Figure 4-11: Arria 10 Gen3 x4 Channel Placement

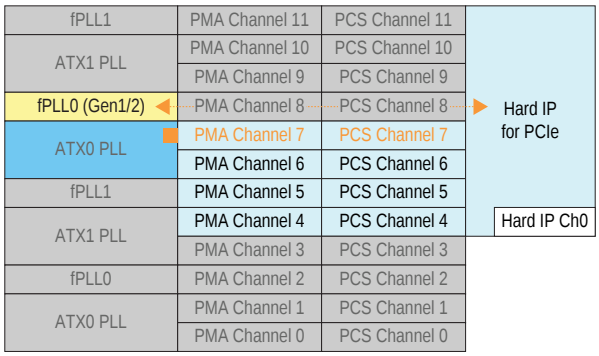
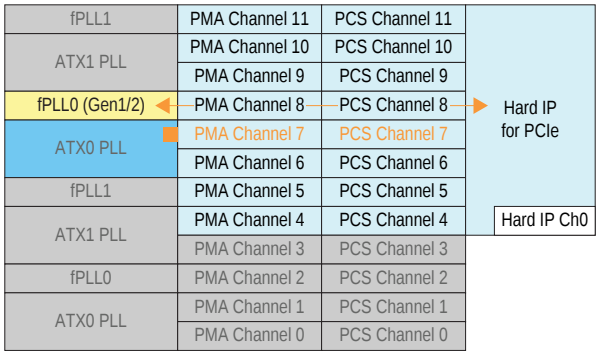


Figure 4-12: Gen3 x8 Channel Placement



IP Core Architecture 5

December 2013

UG-01145_avmm Subscribe Send Feedback

The Arria 10 Hard IP for PCI Express implements the complete PCI Express protocol stack as defined in the *PCI Express Base Specification*. The protocol stack includes the following layers:

- *Transaction Layer*—The Transaction Layer contains the Configuration Space, which manages communication with the Application Layer, 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.

The following figure provides a high-level block diagram of the Arria 10 Hard IP for PCI Express.

Figure 5-1: Arria 10 Hard IP for PCI Express Using the Avalon-MM Interface

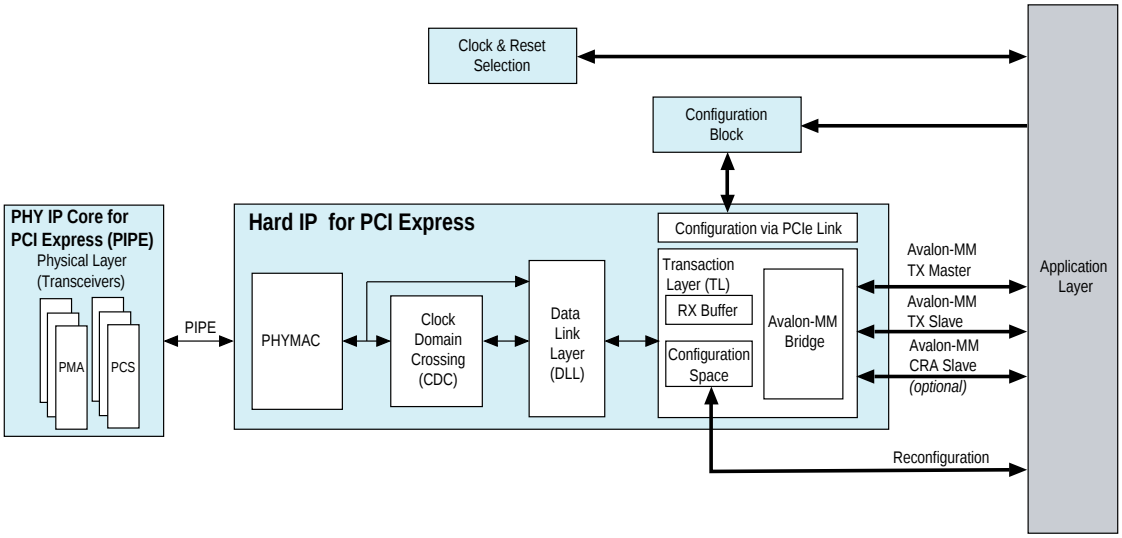


Table 5-1: Application Layer Clock Frequencies

In the current release, variants using the Avalon-MM interface with DMA are only available for Gen2 ×8, and Gen3 ×4 and ×8 configurations.

Lanes	Gen1	Gen2	Gen3
×1	125 MHz @ 64 bits or	125 MHz @ 64 bits	125 MHz @64 bits
×4	125 MHz @ 64 bits	250 MHz @ 64 bits or 125 MHz @ 128 bits	250 MHz @ 128 bits or 125 MHz @ 256 bits
×8	250 MHz @ 64 bits or 125 MHz @ 128 bits	250 MHz @ 128 bits or 125 MHz @ 256 bits	250 MHz @ 256 bits

Related Information

PCI Express Base Specification 3.0

Top-Level Interfaces

Avalon-MM Interface

An Avalon-MM interface connects the Application Layer and the Transaction Layer. The Avalon-MM interface implement the Avalon-MM protocol described in the *Avalon Interface Specifications*. Refer to this specification for information about the Avalon-MM protocol, including timing diagrams.

Related Information

- **64-, 128-, or 256-Bit Avalon-MM Interfaces to the Application Layer** on page 6-2

- **Avalon Interface Specifications**

Clocks and Reset

The *PCI Express Base Specification* requires an input reference clock, which is called `refclk` in this design. The *PCI Express Base Specification* stipulates that the frequency of this clock be 100 MHz.

The *PCI Express Base Specification* also requires a system configuration time of 100 ms. To meet this specification, the Arria 10 Hard IP for PCI Express includes an embedded hard reset controller. This reset controller exits the reset state after the I/O ring of the device is initialized.

Related Information

- **Reset and Clocks** on page 8-1
- **Clock Signals** on page 6-7
- **Reset Signals, Status, and Link Training Signals** on page 6-7

Hard IP Reconfiguration

The PCI Express reconfiguration bus allows you to dynamically change the `read-only` values stored in the Configuration Registers.

Related Information

Hard IP Reconfiguration Interface on page 6-9

Interrupts

The Arria 10 Hard IP for PCI Express offers the following 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.

Related Information

- **Interrupts for Endpoints when Multiple MSI/MSI-X Support Is Enabled**
- **Interrupts for Root Ports**

PIPE

The PIPE interface implements the Intel-designed PIPE interface specification. You can use this parallel interface to speed simulation; however, you cannot use the PIPE interface in actual hardware.

- The Gen1, Gen2, and Gen3 simulation models support pipe and serial simulation.
- For Gen3 simulation, the Altera testbench bypasses Phase 2 and Phase 3 equalization.

Related Information

PIPE Interface Signals on page 6-13

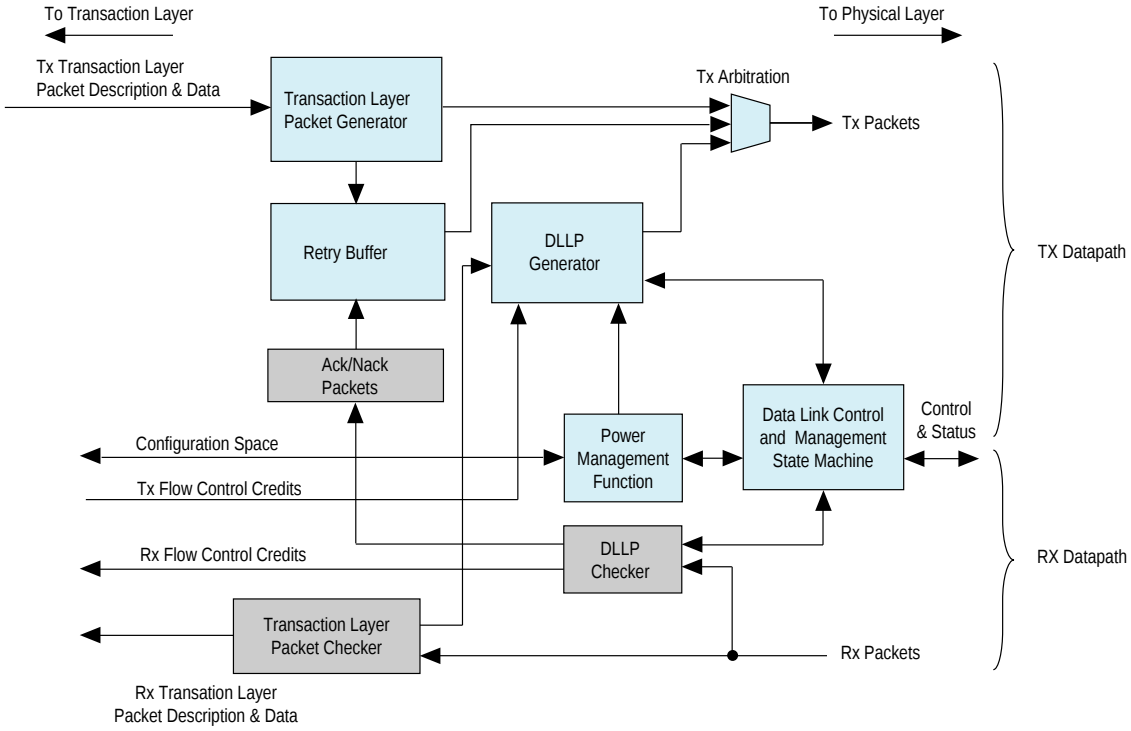
Data Link Layer

The Data Link Layer is located between the Transaction Layer and the Physical Layer. It maintains packet integrity and 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 5-2: Data Link Layer



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 is also connected to the Configuration Space Registers. It initializes the link and flow control credits and reports status to the Configuration Space.
- Power Management—This function handles the handshake to enter low power mode. Such a transition is based on register values in the Configuration Space and received Power Management (PM) DLLPs.
- 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:
- Initialize FC Data Link Layer packet
- ACK/NAK DLLP (high priority)
- Update FC DLLP (high priority)
- PM DLLP
- Retry buffer TLP
- TLP
- Update FC DLLP (low priority)
- ACK/NAK FC DLLP (low priority)

Physical Layer

The Physical Layer is the lowest level of the Arria 10 Hard IP for PCI Express. It is the layer closest to the serial 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, at 2.5 or 5.0 Gbps for Gen2 implementations, and at 2.5, 5.0 or 8.0 Gbps for Gen 3 implementations.

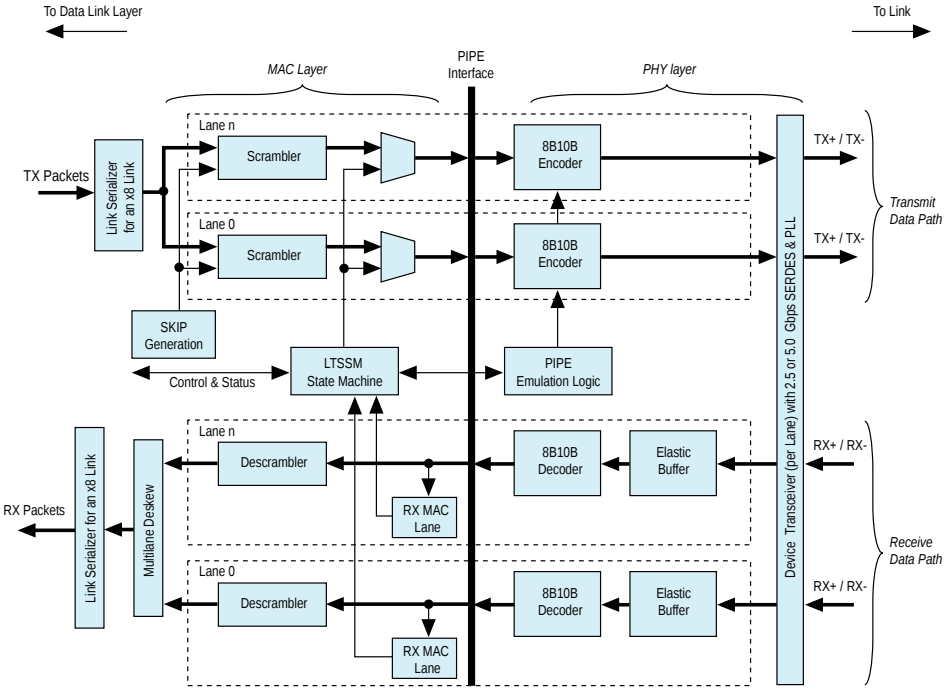
The Physical Layer is responsible for the following actions:

- Initializing the link
- Scrambling/descrambling and 8B/10B encoding/decoding of 2.5 Gbps (Gen1), 5.0 Gbps (Gen2), or 128b/130b encoding/decoding of 8.0 Gbps (Gen3) per lane
- Serializing and deserializing data
- Operating the PIPE 3.0 Interface
- Implementing auto speed negotiation (Gen2 and Gen3)
- Transmitting and decoding the training sequence
- Providing hardware autonomous speed control

- Implementing auto lane reversal

The following figure illustrates the Physical Layer architecture.

Figure 5-3: Physical Layer



The Physical Layer is subdivided by the PIPE Interface Specification into two layers (bracketed horizontally in above figure):

- 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 and 128b/130b 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 10 Hard IP for PCI Express compiles 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. 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.

32-Bit PCI Express Avalon-MM Bridge

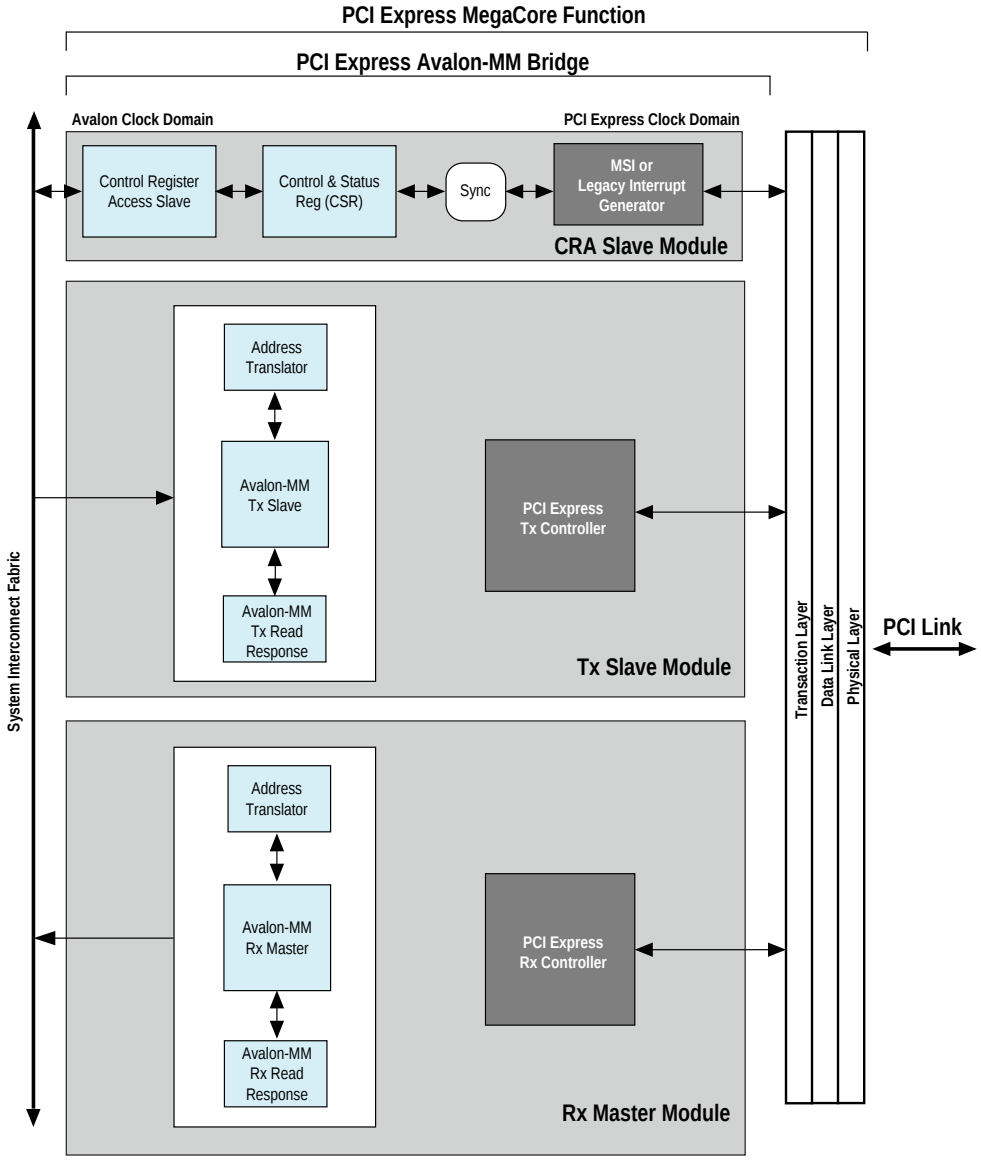
The Avalon-MM Arria 10Hard IP for PCI Express, an Avalon-MM bridge module connects the PCI Express link to the interconnect fabric. The bridge facilitates the design of Endpoints and Root Ports that include Qsys components.

The full-featured Avalon-MM bridge provides three possible Avalon-MM ports: a bursting master, an optional bursting slave, and an optional non-bursting slave. The Avalon-MM bridge comprises the following three modules:

- TX Slave Module—This optional 64- or 128-bit bursting, Avalon-MM dynamic addressing slave port propagates read and write requests of up to 4 KBytes in size from the interconnect fabric to the PCI Express link. The bridge translates requests from the interconnect fabric to PCI Express request packets.
- RX Master Module—This 64- or 128-bit bursting Avalon-MM master port propagates PCI Express requests, converting them to bursting read or write requests to the interconnect fabric.
- Control Register Access (CRA) Slave Module—This optional, 32-bit Avalon-MM dynamic addressing slave port provides access to internal control and status registers from upstream PCI Express devices and external Avalon-MM masters. Implementations that use MSI or dynamic address translation require this port. The CRA port supports single dword read and write requests. It does not support bursting.

When you select the **Single dword completer** in the GUI for the Avalon-MM Hard IP for PCI Express, Qsys substitutes a unpipelined, 32-bit RX master port for the 64- or 128-bit full-featured RX master port. The following figure shows the block diagram of a full-featured PCI Express Avalon-MM bridge.

Figure 5-4: PCI Express Avalon-MM Bridge



The bridge has the following additional characteristics:

- Type 0 and Type 1 vendor-defined incoming messages are discarded
- Completion-to-a-flush request is generated, but not propagated to the interconnect fabric

For End Points, each PCI Express base address register (BAR) in the Transaction Layer maps to a specific, fixed Avalon-MM address range. You can use separate BARs to map to various Avalon-MM slaves connected to the RX Master port. In contrast to Endpoints, Root Ports do not perform any BAR matching and forwards the address to a single RX Avalon-MM master port.

Related Information

Avalon-MM RX Master Block on page 5-17

Avalon-MM Bridge TLPs

The PCI Express to Avalon-MM bridge translates the PCI Express read, write, and completion Transaction Layer Packets (TLPs) into standard Avalon-MM read and write commands typically used by master and slave interfaces. This PCI Express to Avalon-MM bridge also translates Avalon-MM read, write and read data commands to PCI Express read, write and completion TLPs. The following topics describe the Avalon-MM bridges translations.

Avalon-MM-to-PCI Express Write Requests

The Avalon-MM bridge accepts Avalon-MM burst write requests with a burst size of up to 512 Bytes at the Avalon-MM TX slave interface. The Avalon-MM bridge converts the write requests to one or more PCI Express write packets with 32– or 64-bit addresses based on the address translation configuration, the request address, and the maximum payload size.

The Avalon-MM write requests can start on any address in the range defined in the PCI Express address table parameters. The bridge splits incoming burst writes that cross a 4 KByte boundary into at least two separate PCI Express packets. The bridge also considers the root complex requirement for maximum payload on the PCI Express side by further segmenting the packets if needed.

The bridge requires Avalon-MM write requests with a burst count of greater than one to adhere to the following byte enable rules:

- The Avalon-MM byte enables must be asserted in the first qword of the burst.
- All subsequent byte enables must be asserted until the deasserting byte enable.
- The Avalon-MM byte enables may deassert, but only in the last qword of the burst.

Note: To improve PCI Express throughput, Altera recommends using an Avalon-MM burst master without any byte-enable restrictions.

Avalon-MM-to-PCI Express Upstream Read Requests

The PCI Express Avalon-MM bridge converts read requests from the system interconnect fabric to PCI Express read requests with 32-bit or 64-bit addresses based on the address translation configuration, the request address, and the maximum read size.

The Avalon-MM TX slave interface of a PCI Express Avalon-MM bridge can receive read requests with burst sizes of up to 512 bytes sent to any address. However, the bridge limits read requests sent to the PCI Express link to a maximum of 256 bytes. Additionally, the bridge must prevent each PCI Express read request packet from crossing a 4 KByte address boundary. Therefore, the bridge may split an Avalon-MM read request into multiple PCI Express read packets based on the address and the size of the read request.

Avalon-MM bridge supports up to eight outstanding reads from Avalon-MM interface. Once the bridge has eight outstanding read requests, the `TxsWaitRequest` signal is asserted to block additional read requests. When a read request completes, the Avalon-MM bridge can accept another request.

For Avalon-MM read requests with a burst count greater than one, all byte enables must be asserted. There are no restrictions on byte enables for Avalon-MM read requests with a burst count of one. An invalid Avalon-MM request can adversely affect system functionality, resulting in a completion with the abort status set. An example of an invalid request is one with an incorrect address.

PCI Express-to-Avalon-MM Read Completions

The PCI Express Avalon-MM bridge returns read completion packets to the initiating Avalon-MM master in the issuing order. The bridge supports multiple and out-of-order completion packets.

PCI Express-to-Avalon-MM Downstream Write Requests

The PCI Express Avalon-MM bridge receives PCI Express write requests, it converts them to burst write requests before sending them to the interconnect fabric. For Endpoints, the bridge translates the PCI Express address to the Avalon-MM address space based on the BAR hit information and on address translation table values configured during the IP core parameterization. For Root Ports, all requests are forwarded to a single RX Avalon-MM master that drives them to the interconnect fabric. Malformed write packets are dropped, and therefore do not appear on the Avalon-MM interface.

For downstream write and read requests, if more than one byte enable is asserted, the byte lanes must be adjacent. In addition, the byte enables must be aligned to the size of the read or write request.

As an example, the following table lists the byte enables for 32-bit data.

Table 5-2: Valid Byte Enable Configurations

Byte Enable Value	Description
4'b1111	Write full 32 bits
4'b0011	Write the lower 2 bytes
4'b1100	Write the upper 2 bytes
4'b0001	Write byte 0 only
4'b0010	Write byte 1 only
4'b0100	Write byte 2 only
4'b1000	Write byte 3 only

In burst mode, the Arria 10 Hard IP for PCI Express supports only byte enable values that correspond to a contiguous data burst. For the 32-bit data width example, valid values in the first data phase are 4'b1111, 4'b1110, 4'b1100, and 4'b1000, and valid values in the final data phase of the burst are 4'b1111, 4'b0111, 4'b0011, and 4'b0001. Intermediate data phases in the burst can only have byte enable value 4'b1111.

PCI Express-to-Avalon-MM Downstream Read Requests

The PCI Express Avalon-MM bridge sends PCI Express read packets to the interconnect fabric as burst reads with a maximum burst size of 512 bytes. For Endpoints, the bridge converts the PCI Express address to the Avalon-MM address space based on the BAR hit information and address translation lookup table values. The RX Avalon-MM master port drives the received address to the fabric. You can set up the Address Translation Table Configuration in the GUI. Unsupported read requests generate a completer abort response.

Related Information

Minimizing BAR Sizes and the PCIe Address Space on page 5-12

Avalon-MM-to-PCI Express Read Completions

The PCI Express Avalon-MM bridge converts read response data from Application Layer Avalon-MM slaves to PCI Express completion packets and sends them to the Transaction Layer.

A single read request may produce multiple completion packets based on the **Maximum payload size** and the size of the received read request. For example, if the read is 512 bytes but the **Maximum payload size** 128 bytes, the bridge produces four completion packets of 128 bytes each. The bridge does not generate out-of-order completions. You can specify the **Maximum payload size** parameter on the **Device** tab under the **PCI Express/PCI Capabilities** heading in the GUI.

Related Information

Device Capabilities on page 3-6

PCI Express-to-Avalon-MM Address Translation for 32-Bit Bridge

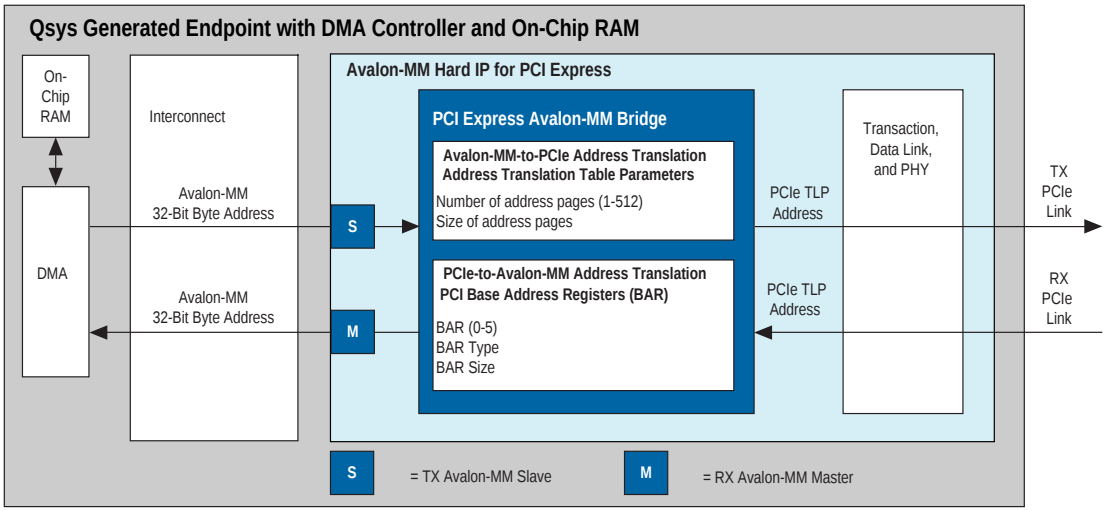
The PCI Express Avalon-MM Bridge translates the system-level physical addresses, typically up to 64 bits, to the significantly smaller addresses required by the Application Layer's Avalon-MM slave components.

Note: Starting with the 13.0 version of the Quartus II software, the PCI Express-to-Avalon-MM bridge supports both 32- and 64-bit addresses. If you select 64-bit addressing the bridge does not perform address translation. It drives the addresses specified to the interconnect fabric. You can limit the number of address bits used by Avalon-MM slave components to the actual size required by specifying the address size in the Avalon-MM slave component GUI.

You can specify up to six BARs for address translation when you customize your Hard IP for PCI Express as described in *Base Address Register (BAR) and Expansion ROM Settings*. When 32-bit addresses are specified, the PCI Express Avalon-MM Bridge also translates Application Layer addresses to system-level physical addresses as described in *Avalon-MM-to-PCI Express Address Translation Algorithm for 32-Bit Addressing*.

The following figure provides a high-level view of address translation in both directions.

Figure 5-5: Address Translation in TX and RX Directions For Endpoints



Note: When configured as a Root Port, a single RX Avalon-MM master forwards all RX TLPs to the Qsys interconnect.

The Avalon-MM RX master module port has an 8-byte datapath in 64-bit mode and a 16-byte datapath in 128-bit mode. The Qsys interconnect fabric manages mismatched port widths transparently.

As Memory Request TLPs are received from the PCIe link, the most significant bits are used in the BAR matching as described in the PCI specifications. The least significant bits not used in the BAR match process are passed unchanged as the Avalon-MM address for that BAR's RX Master port.

For example, consider the following configuration specified using the Base Address Registers in the GUI.

1. BAR1:0 is a **64-bit prefetchable memory** that is **4KBytes -12 bits**
2. System software programs BAR1:0 to have a base address of0x00001234 56789000
3. A TLP received with address 0x00001234 56789870
4. The upper 52 bits (0x0000123456789) are used in the BAR matching process, so this request matches.
5. The lower 12 bits, 0x870, are passed through as the Avalon address on the Rxm_BAR0 Avalon-MM Master port. The BAR matching software replaces the upper 20 bits of the address with the Avalon-MM base address.

Related Information

- **Base Address Register (BAR) and Expansion ROM Settings** on page 3-4
- **Avalon-MM-to-PCI Express Address Translation Algorithm for 32-Bit Addressing** on page 5-14

Minimizing BAR Sizes and the PCIe Address Space

For designs that include multiple BARs, you may need to modify the base address assignments auto-assigned by Qsys in order to minimize the address space that the BARs consume. For example, consider a Qsys system with the following components:

- **Offchip_D ata_Mem DDR3** (SDRAM Controller with UniPHY) controlling 256 MBytes of memory—Qsys auto-assigned a base address of 0x00000000
- **Quick_Data_Mem** (On-Chip Memory (RAM or ROM)) of 4 KBytes—Qsys auto-assigned a base address of 0x10000000
- **Instruction_Mem** (On-Chip Memory (RAM or ROM)) of 64 KBytes—Qsys auto-assigned a base address of 0x10020000
- **PCIe** (Avalon-MM Arria 10 Hard IP for PCI Express)
  - **Cra** (Avalon-MM Slave)—auto assigned base address of 0x10004000
  - **Rxm_BAR0** connects to **Offchip_Data_Mem DD R3 avl**
  - **Rxm_BAR2** connects to **Quick_Data_Mem s1**
  - **Rxm_BAR4** connects to PCIe. **Cra Avalon MM Slave**
- **Nios2** (Nios® II Processor)
  - **data_master** connects to **PCIe Cra**, **Offchip_Data_Mem DDR3 avl**, **Quick_Data_Mem s1**, **Instruction_Mem s1**, **Nios2 jtag_debug_mod ule**
  - **instruction_master** connects to **Instruction_Mem s1**

Figure 5-6: Qsys System for PCI Express with Poor Address Space Utilization

The following figure illustrates this Qsys system. (This figure uses a filter to hide the Conduit interfaces that are not relevant in this discussion.)

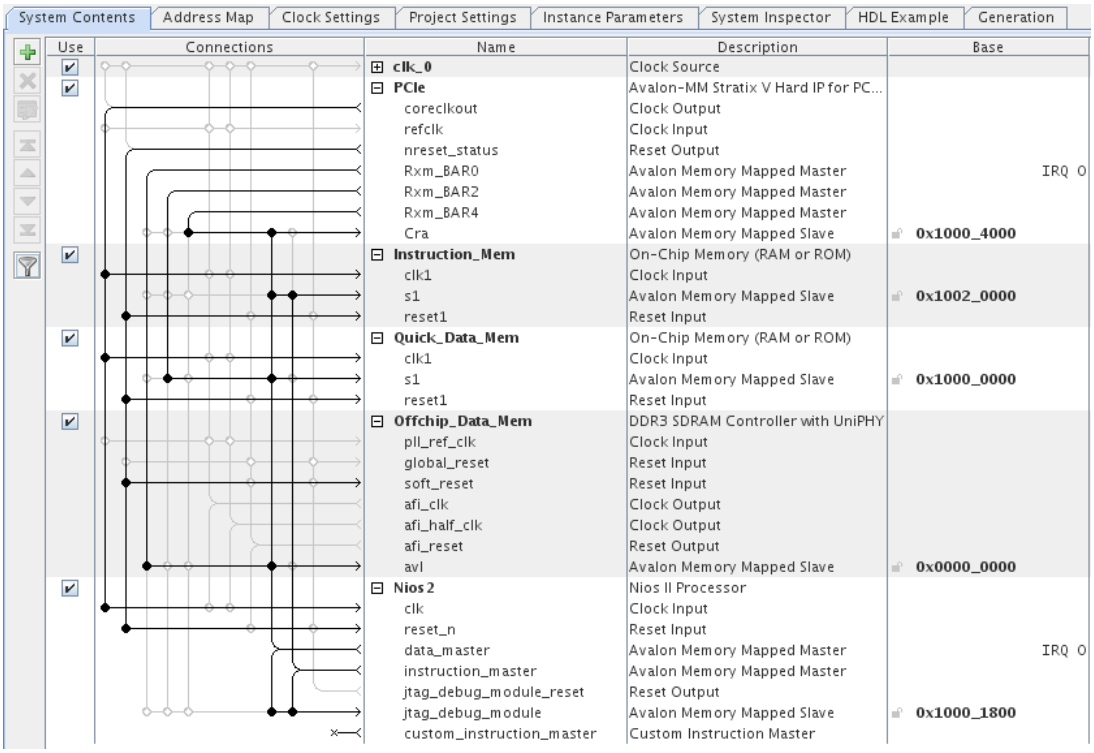
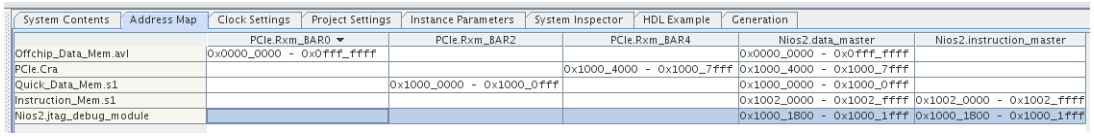


Figure 5-7: Poor Address Map

The following figure illustrates the address map for this system.



The auto-assigned base addresses result in the following three large BARs:

- BAR0 is 28 bits. This is the optimal size because it addresses the **Offchip_Data_Mem** which requires 28 address bits.
- BAR2 is 29 bits. BAR2 addresses the **Quick_Data_Mem** which is 4 KBytes;. It should only require 12 address bits; however, it is consuming 512 MBytes of address space.
- BAR4 is also 29 bits. BAR4 address **PCIe Cra** which is 16 KBytes. It should only require 14 address bits; however, it is also consuming 512 MBytes of address space.

This design is consuming 1.25GB of PCIe address space when only 276 MBytes are actually required. The solution is to edit the address map to place the base address of each BAR at 0x0000_0000. The following figure illustrates the optimized address map.

Figure 5-8: Optimized Address Map

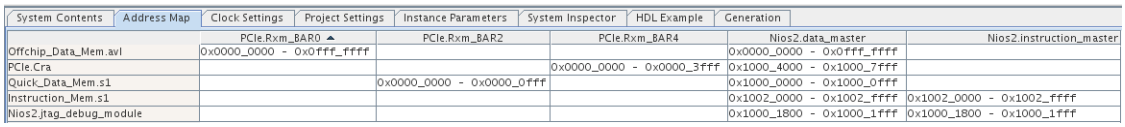


Figure 5-9: Reduced Address Bits for BAR2 and BAR4

The following figure shows the number of address bits required when the smaller memories accessed by BAR2 and BAR4 have a base address of 0x0000_0000.

For cases where the BAR Avalon-MM RX master port connects to more than one Avalon-MM slave, assign the base addresses of the slaves sequentially and place the slaves in the smallest power-of-two-sized address space possible. Doing so minimizes the system address space used by the BAR.

Related Information

Address Map Tab (Qsys)

Avalon-MM-to-PCI Express Address Translation Algorithm for 32-Bit Addressing

Note: The PCI Express-to-Avalon-MM bridge supports both 32- and 64-bit addresses. If you select 64-bit addressing the bridge does not perform address translation.

When you specify 32-bit addresses, the Avalon-MM address of a received request on the TX Avalon-MM slave port is translated to the PCI Express address before the request packet is sent to the Transaction Layer. You can specify up to 512 address pages and sizes ranging from 4 KByte to 4 GBytes when you customize your Avalon-MM Arria 10 Hard IP for PCI Express as described in "R**Avalon to PCIe Address Translation Settings ###addr_trans###" on page 6–13. This address translation process proceeds by replacing the MSB of the Avalon-MM address with the value from a specific translation table entry; the LSB remains unchanged. The number of MSBs to be replaced is calculated based on the total address space of the upstream PCI Express devices that the Avalon-MM Hard IP for PCI Express can access. The number of MSB bits is defined by the difference between the maximum number of bits required to represent the address space supported by the upstream PCI Express device minus the number of bits required to represent the **Size of address pages** which are the LSB pass-through bits (*N*). The **Size of address pages** (*N*) is applied to all entries in the translation table.

Each of the 512 possible entries corresponds to the base address of a PCI Express memory segment of a specific size. The segment size of each entry must be identical. The total size of all the memory segments is used to determine the number of address MSB to be replaced. In addition, each entry has a 2-bit field, `Sp[1:0]`, that specifies 32-bit or 64-bit PCI Express addressing for the translated address. The most significant bits of the Avalon-MM address are used by the interconnect fabric to select the slave port and are not available to the slave. The next most significant bits of the Avalon-MM address index the address translation entry to be used for the translation process of MSB replacement.

For example, if the core is configured with an address translation table with the following attributes:

- **Number of Address Pages—16**
- **Size of Address Pages—1 MByte**
- **PCI Express Address Size—64 bits**

then the values in the following figure are:

- $N = 20$ (due to the 1 MByte page size)
- $Q = 16$ (number of pages)
- $M = 24$ (20 + 4 bit page selection)
- $P = 64$

In this case, the Avalon address is interpreted as follows:

- Bits [31:24] select the TX slave module port from among other slaves connected to the same master by the system interconnect fabric. The decode is based on the base addresses assigned in Qsys.
- Bits [23:20] select the address translation table entry.
- Bits [63:20] of the address translation table entry become PCI Express address bits [63:20].
- Bits [19:0] are passed through and become PCI Express address bits [19:0].

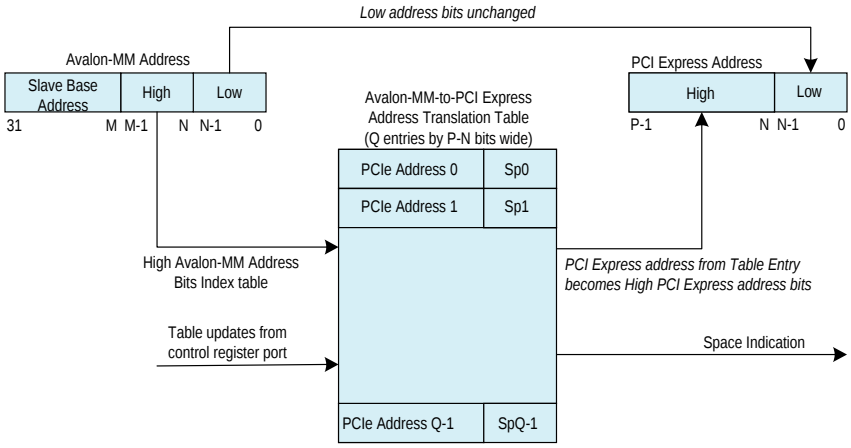
The address translation table is dynamically configured at run time. The address translation table is implemented in memory and can be accessed through the CRA slave module. Dynamic configuration is optimal in a typical PCI Express system where address allocation occurs after BIOS initialization.

For more information about how to access the dynamic address translation table through the CRA slave, refer to the "Avalon-MM-to-PCI Express Address Translation Table 0x1000–0x1FFF" on page 9–17.

Figure 5-10: Avalon-MM-to-PCI Express Address Translation

The following figure depicts the Avalon-MM-to-PCI Express address translation process. In this figure the variables represent the following paramers:

- N—the number of pass-through bits.
- M—the number of Avalon-MM address bits.
- P—the number of PCIe address bits.
- Q—the number of translation table entries.
- `Sp[1:0]`—the space indication for each entry.



Completer Only Single Dword Endpoint

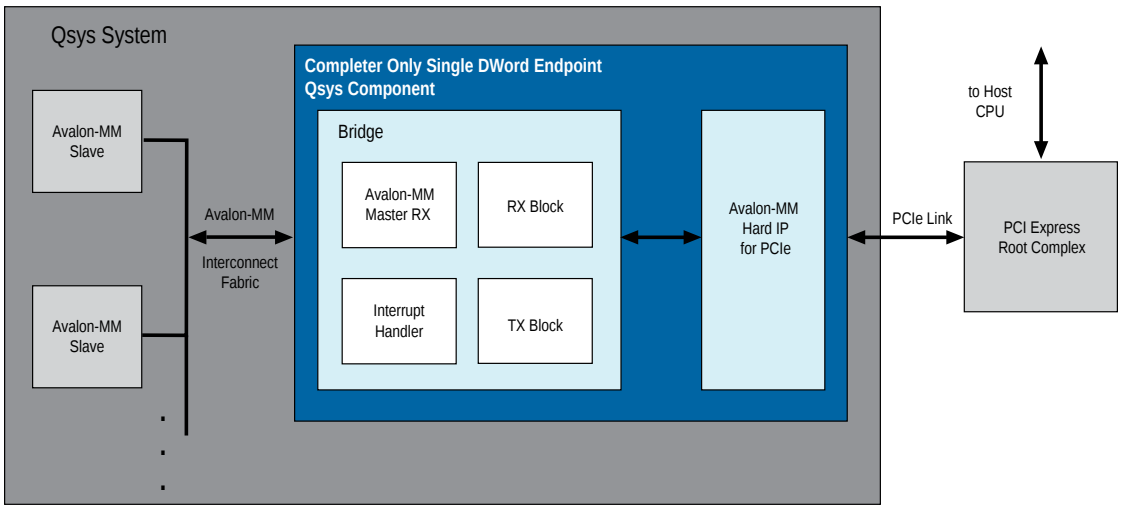
The completer only single dword endpoint is intended for applications that use the PCI Express protocol to perform simple read and write register accesses from a host CPU. The completer only single dword endpoint is a hard IP implementation available for Qsys systems, and includes an Avalon-MM interface to the Application Layer. The Avalon-MM interface connection in this variation is 32 bits wide. This endpoint is not pipelined; at any time a single request can be outstanding.

The completer-only single dword endpoint supports the following requests:

- Read and write requests of a single dword (32 bits) from the Root Complex
- Completion with Completer Abort status generation for other types of non-posted requests
- INTX or MSI support with one Avalon-MM interrupt source

The following figure shows a Qsys system that includes a completer-only single dword endpoint.

Figure 5-11: Qsys Design Including Completer Only Single Dword Endpoint for PCI Express



As the above figure shows, the completer-only single dword endpoint connects to a PCI Express root complex. A bridge component includes the Arria 10 Hard IP for PCI Express TX and RX blocks, an Avalon-MM RX master, and an interrupt handler. The bridge connects to the FPGA fabric using an Avalon-MM interface. The following sections provide an overview of each block in the bridge.

RX Block

The RX Block control logic interfaces to the hard IP block to process requests from the root complex. It supports memory reads and writes of a single dword. It generates a completion with Completer Abort (CA) status for read requests greater than four bytes and discards all write data without further action for write requests greater than four bytes.

The RX block passes header information to the Avalon-MM master, which generates the corresponding transaction to the Avalon-MM interface. The bridge accepts no additional requests while a request is being processed. While processing a read request, the RX block deasserts the `ready` signal until the TX block

sends the corresponding completion packet to the hard IP block. While processing a write request, the RX block sends the request to the Avalon-MM interconnect fabric before accepting the next request.

Avalon-MM RX Master Block

The 32-bit Avalon-MM master connects to the Avalon-MM interconnect fabric. It drives read and write requests to the connected Avalon-MM slaves, performing the required address translation. The RX master supports all legal combinations of byte enables for both read and write requests.

For more information about legal combinations of byte enables, refer to *Chapter 3, Avalon Memory Mapped Interfaces* in the Avalon Interface Specifications.

Related Information

Avalon Interface Specifications

TX Block

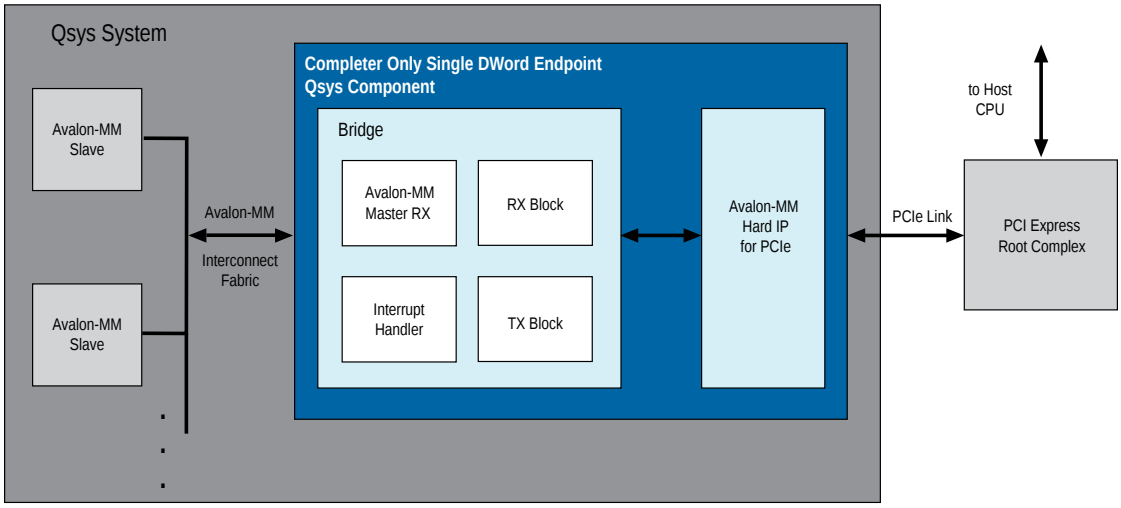
The TX block sends completion information to the Avalon-MM Hard IP for PCI Express which sends this information to the root complex. The TX completion block generates a completion packet with Completer Abort (CA) status and no completion data for unsupported requests. The TX completion block also supports the zero-length read (flush) command.

Interrupt Handler Block

The interrupt handler implements both INTX and MSI interrupts. The `msi_enable` bit in the configuration register specifies the interrupt type. The `msi_enable_bit` is part of the MSI message control portion in the MSI Capability structure. It is bit[16] of address 0x050 in the Configuration Space registers. If the `msi_enable` bit is on, an MSI request is sent to the Arria 10 Hard IP for PCI Express when received, otherwise INTX is signaled. The interrupt handler block supports a single interrupt source, so that software may assume the source. You can disable interrupts by leaving the interrupt signal unconnected in the IRQ column of Qsys.

When the MSI registers in the Configuration Space of the Completer Only Single Dword Arria 10 Hard IP for PCI Express are updated, there is a delay before this information is propagated to the Bridge module shown in the following figure.

Figure 5-12: Qsys Design Including Completer Only Single Dword Endpoint for PCI Express



You must allow time for the Bridge module to update the MSI register information. Normally, setting up MSI registers occurs during enumeration process. Under normal operation, initialization of the MSI registers should occur substantially before any interrupt is generated. However, failure to wait until the update completes may result in any of the following behaviors:

- Sending a legacy interrupt instead of an MSI interrupt
- Sending an MSI interrupt instead of a legacy interrupt
- Loss of an interrupt request

According to the *PCI Express Base Specification*, if `MSI_enable`=0 and the `Disable Legacy Interrupt bit`=1 in the Configuration Space `Command` register (0x004), the Hard IP should not send legacy interrupt messages when an interrupt is generated.

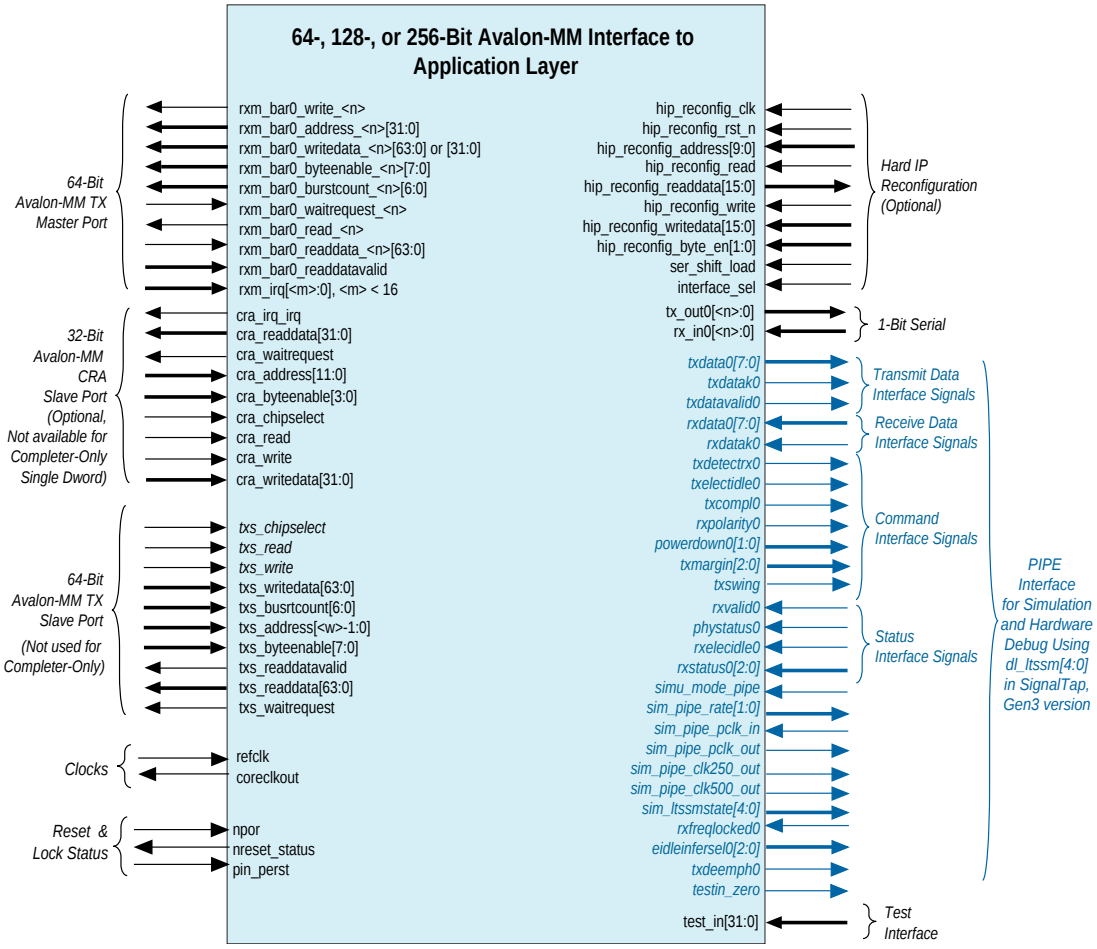
IP Core Interfaces 6

Subscribe Send Feedback

This chapter describes the top-level signals of the Arria 10 Hard IP for PCI Express using the Avalon-MM interface to the Application Layer. The Avalon-MM bridge translates PCI Express read, write and completion TLPs into standard Avalon-MM read and write commands for the RX interface. For the TX interface, the bridge translates Avalon-MM reads and writes into PCI Express TLPs. The Avalon-MM reads and write commands are the same as those used by master and slave interfaces to access memories and registers. Consequently, you do not need a detailed understanding of the PCI Express TLPs to use this Avalon-MM variant. This variant is available for Gen 1 and Gen2 x1, x2, x4, and x8 and Gen3 x1 and x4 Endpoints and Root Ports.

64-, 128-, or 256-Bit Avalon-MM Interfaces to the Application Layer

Figure 6-1: Signals in 64-, 128-, or 256-Bit Avalon-MM Interface to the Application Layer



Note: In this figure, signals listed for `rxm_bar0` are also exist for `rxm_bar1` through `rxm_bar5` when those BARs are enabled in the parameter editor.

Variations using the Avalon-MM interface implement the Avalon-MM protocol described in the *Avalon Interface Specifications*. Refer to this specification for information about the Avalon-MM protocol, including timing diagrams.

Related Information

Avalon Interface Specifications

RX Avalon-MM Master Signals

This Avalon-MM master port propagates PCI Express requests to the Qsys interconnect fabric. For the full-feature IP core it propagates requests as bursting reads or writes. A separate Avalon-MM master port corresponds to each BAR. Signals that include lane number 0 also exist for BAR1–BAR5 when additional BARs are enabled. The following table lists the RX Master interface ports.

Table 6-1: Avalon-MM RX Master Interface Signals

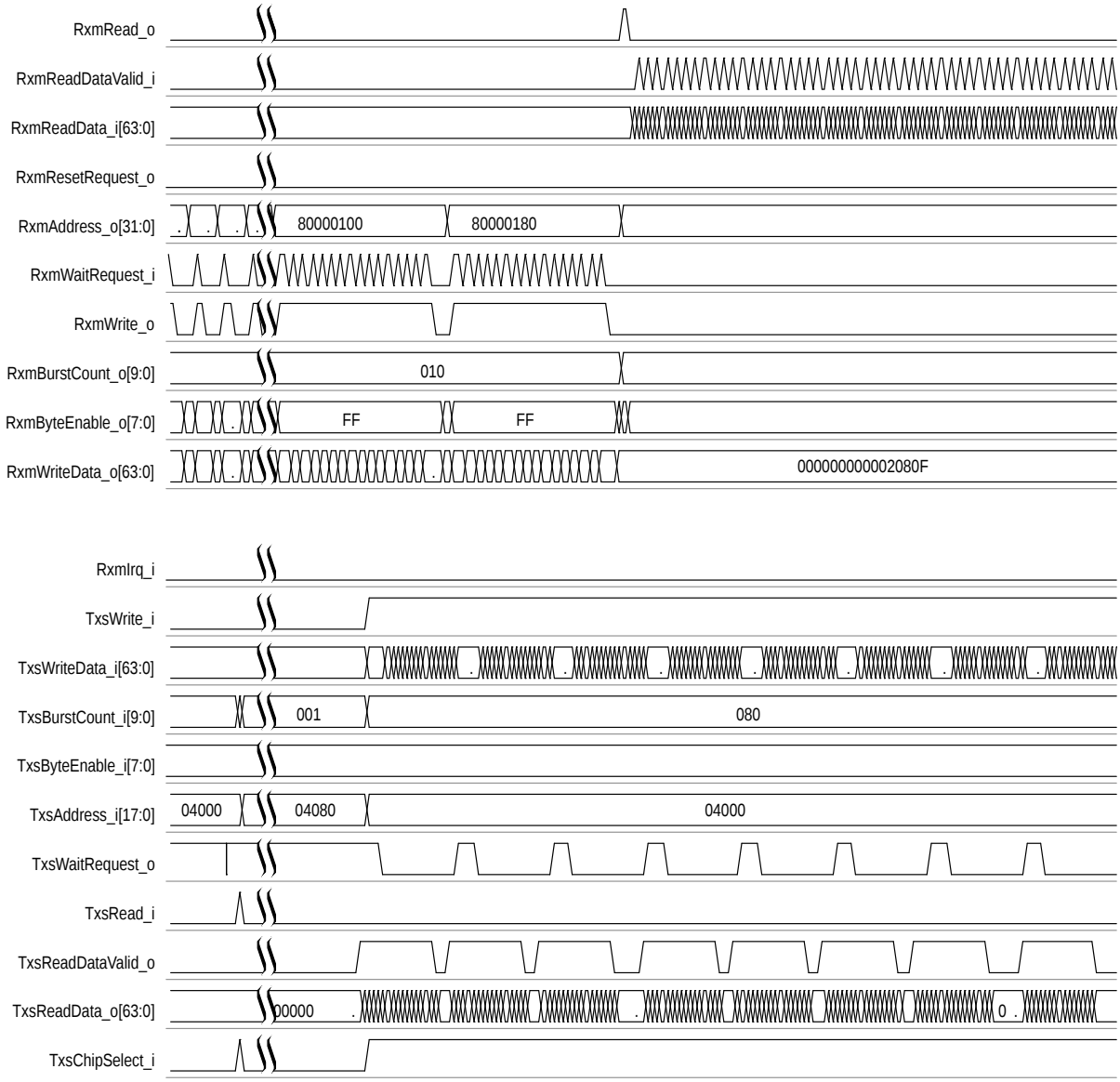
Signal Name	I/O	Description
rxm_bar0_write_<n> (1)	O	Asserted by the core to request a write to an Avalon-MM slave.
rxm_bar0_address_<*n*>[31:0]	O	The address of the Avalon-MM slave being accessed.
rxm_bar0_writedata_<*n*>[<*w*>-1:0]	O	RX data being written to slave. <*w*> = 64 for the full-featured IP core. <*w*> = 32 for the completer-only IP core.
rxm_bar0_byteenable_<*n*>[<*w*>-1:0]	O	Byte enable for write data.
rxm_bar0_burstcount_<*n*>[6:0]	O	The burst count, measured in qwords, of the RX write or read request. The width indicates the maximum data that can be requested. The maximum data in a burst is 512 bytes.
rxm_bar0_waitrequest_<*n*>	I	Asserted by the external Avalon-MM slave to hold data transfer.
rxm_bar0_read_<*n*>	O	Asserted by the core to request a read.
rxm_bar0_readdata_<*n*>[<*w*>-1:0]	I	Read data returned from Avalon-MM slave in response to a read request. This data is sent to the IP core through the TX interface. <*w*> = 64 for the full-featured IP core. <*w*> = 32 for the completer-only IP core.
rxm_bar0_readdatavalid_<*n*>	I	Asserted by the system interconnect fabric to indicate that the read data on is valid.
rxm_irq_<n>[<*m*>:0]	I	Indicates an interrupt request asserted from the system interconnect fabric. This signal is only available when the CRA port is enabled. Qsys-generated variations have as many as 16 individual interrupt signals (<*m*>≤15). If rxm_irq_<n>[<*m*>:0] is asserted on consecutive cycles without the deassertion of all interrupt inputs, no MSI message is sent for subsequent interrupts. To avoid losing interrupts, software must ensure that all interrupt sources are cleared for each MSI message received.

Note:

1. <*n*>represents the BAR number for all signals. The core supports up to 6 BARs.

The following figure illustrates the RX master port propagating requests to the Application Layer and also shows simultaneous, DMA read and write activity

Figure 6-2: Simultaneous DMA Read, DMA Write, and Target Access



32-Bit Non-Bursting Avalon-MM Control Register Access (CRA) Slave Signals

The optional CRA port for the full-featured IP core allows upstream PCI Express devices and external Avalon-MM masters to access internal control and status registers.

The following table describes the CRA slave signals.

Table 6-2: Avalon-MM CRA Slave Interface Signals

Signal Name	I/O	Type	Description
`cra_irq_irq`	O	Irq	Interrupt request. A port request for an Avalon-MM interrupt.

Signal Name	I/O	Type	Description
cra_readdata[31:0]	O	Readdata	Read data lines
cra_waitrequest	O	Waitrequest	Wait request to hold off more requests
cra_address[11:0]	I	Address	An address space of 16,384 bytes is allocated for the control registers. Avalon-MM slave addresses provide address resolution down to the width of the slave data bus. Because all addresses are byte addresses, this address logically goes down to bit 2. Bits 1 and 0 are 0.
cra_byteenable[3:0]	I	Byteenable	Byte enable
cra_chipselect	I	Chipselect	Chip select signal to this slave
cra_read	I	Read	Read enable
cra_write	I	Write	Write request
cra_writedata[31:0]	I	Writedata	Write data

64-Bit Bursting TX Avalon-MM Slave Signals

This optional Avalon-MM bursting slave port propagates requests from the interconnect fabric to the full-featured Avalon-MM Arria 10 Hard IP for PCI Express. Requests from the interconnect fabric are translated into PCI Express request packets. Incoming requests can be up to 512 bytes. For better performance, Altera recommends using smaller read request size (a maximum of 512 bytes).

The following table lists the TX slave interface signals.

Table 6-3: Avalon-MM TX Slave Interface Signals

Signal Name	I/O	Description
txs_chipselect	I	The system interconnect fabric asserts this signal to select the TX slave port.
txs_read	I	Read request asserted by the system interconnect fabric to request a read.

Signal Name	I/O	Description
txs_Write	I	Write request asserted by the system interconnect fabric to request a write. The Avalon-MM Arria 10 Hard IP for PCI Express requires that the Avalon-MM master assert this signal continuously from the first data phase through the final data phase of the burst. The Avalon-MM master Application Layer must guarantee the data can be passed to the interconnect fabric with no pauses. This behavior is most easily implemented with a store and forward buffer in the Avalon-MM master.
txs_writedata[63:0]	I	Write data sent by the external Avalon-MM master to the TX slave port.
txs_burstcount[6:0]	I	Asserted by the system interconnect fabric indicating the amount of data requested. The count unit is the amount of data that is transferred in a single cycle, that is, the width of the bus. The burst count is limited to 512 bytes.
txs_address[<*w*>-1:0]	I	Address of the read or write request from the external Avalon-MM master. This address translates to 64-bit or 32-bit PCI Express addresses based on the translation table. The <*w*> value is determined when the system is created.
txs_byteenable[<*w*>:0]	I	Write byte enable for data. A burst must be continuous. Therefore all intermediate data phases of a burst must have a byte enable value of 0xFF. The first and final data phases of a burst can have other valid values.
txs_readdatavalid	O	Asserted by the bridge to indicate that read data is valid.
txs_readdata[63:0]	O	The bridge returns the read data on this bus when the RX read completions for the read have been received and stored in the internal buffer.
txs_waitrequest	O	Asserted by the bridge to hold off write data when running out of buffer space. If this signal is asserted during an operation, the master should maintain the txs_Read signal (or txs_Write signal and txs_WriteData) stable until after txs_WaitRequest is deasserted.

MSI and MSI-X Interfaces

The IP Core supports both MSI and MSI-X interrupts when both are enabled in the GUI. The Application Layer uses the information from this interface to send MSI and MSI-X to the Root Port via the Tx Slave Control Interface

Table 6-4: CRA Slave Interface

Signal Name	I/O	Description
`MsiIntf[81:0]`	Output	This bus provides the following MSI address, data, and enabled signals: • `msi_intf[81]`: Master enable • `msi_intf[80]`: MSI enable • `msi_intf[79:64]`: MSI data • `msi_intf[63:0]`: MSI address
`MsixIntfc_o[15:0]`	Output	Message Control Register for MSI-X as defined *Section 6.8.2* of the *PCI Local* Bus Specification, Rev. 3.0.

Related Information

PCI Local Bus Specification, Rev. 3.0

Clock Signals

Table 6-5: Clock Signals Hard IP Implementation

Signal	I/O	Description
`refclk`	I	Reference clock for the Arria 10 Hard IP for PCI Express. It must have the frequency specified under the **System Settings** heading in the parameter editor.
`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 *coreclkout_hip Values for All Parameterizations* in the *Reset and Clocks* chapter .

Related Information

Clocks on page 8-4

Reset Signals, Status, and Link Training Signals

The following table describes the reset signals. Refer to *Chapter 10, Reset and Clocks* for more information about the reset sequence and a block diagram of the reset logic. The following table describes reset signals used in all IP Cores for PCI Express.

Table 6-6: Reset Signals

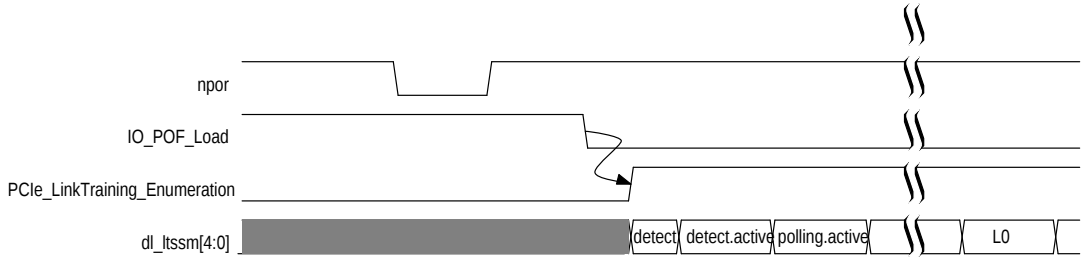
Signal	I/O	Description
npor	I	Active low reset signal. In the Altera hardware example designs, npor is the OR of pin_perst and local_rstn coming from the software Application Layer. If you do not drive a soft reset signal from the Application Layer, this signal must be derived from pin_perst. You cannot disable this signal. Asynchronous. Resets the entire Arria 10 Hard IP for PCI Express IP Core and transceiver. In systems that use the hard reset controller, this signal is *edge*, *not level* sensitive; consequently, you cannot use a low value on this signal to hold custom logic in reset. For more information about the hard and soft reset controllers, refer to Reset.
nreset_status	O	Active low reset signal. apps_rstn, which is derived from npor or pin_perstn.
pin_perst	I	Active low reset from the PCIe reset pin of the device. It resets the datapath and control registers. This signal is required for Configuration via Protocol (CvP). For more information about CvP refer to *Configuration via Protocol (CvP)*. Arria 10 devices have up to 4 instances of the Hard IP for PCI Express. Each instance has its own pin_perst signal. Every Arria 10 device has 4 nPE RST pins, even devices with fewer than 4 instances of the Hard IP for PCI Express. *You must connect the* pin_perst *of each Hard IP instance to the corresponding* nPERST **pin of the device.* These pins have the following locations: • nPERSTL0: bottom left Hard IP and CvP blocks • nPERSTL1: top left Hard IP block • nPERSTR0: bottom right Hard IP block • nPERSTR1: top right Hard IP block For example, if you are using the Hard IP instance in the bottom left corner of the device, you must connect pin_perst to nPERSL0. For maximum use of the Arria 10 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 10 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 nPERST* 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.

Signal	I/O	Description
		• The input signal meets the overshoot specification for 100°C operation as defined in the device handbook.

The following table describes additional signals related to the reset function for the Arria 10 Hard IP for PCI Express IP Core that uses the Avalon-ST interface, including the `ltsssm_state[4:0]` bus that indicates the current link training state.

Figure 6-3: Reset and Link Training Timing Relationships

The following figure illustrates the timing relationship between `npor` and the LTSSM L0 state.



Note: To meet the 100 ms system configuration time, you must use the fast passive parallel configuration scheme with and a 32-bit data width (FPP x32).

Related Information

- **Reset** on page 8-1
- **Clocks** on page 8-4
- **PCI Express Card Electromechanical Specification 2.0**

Hard IP Reconfiguration Interface

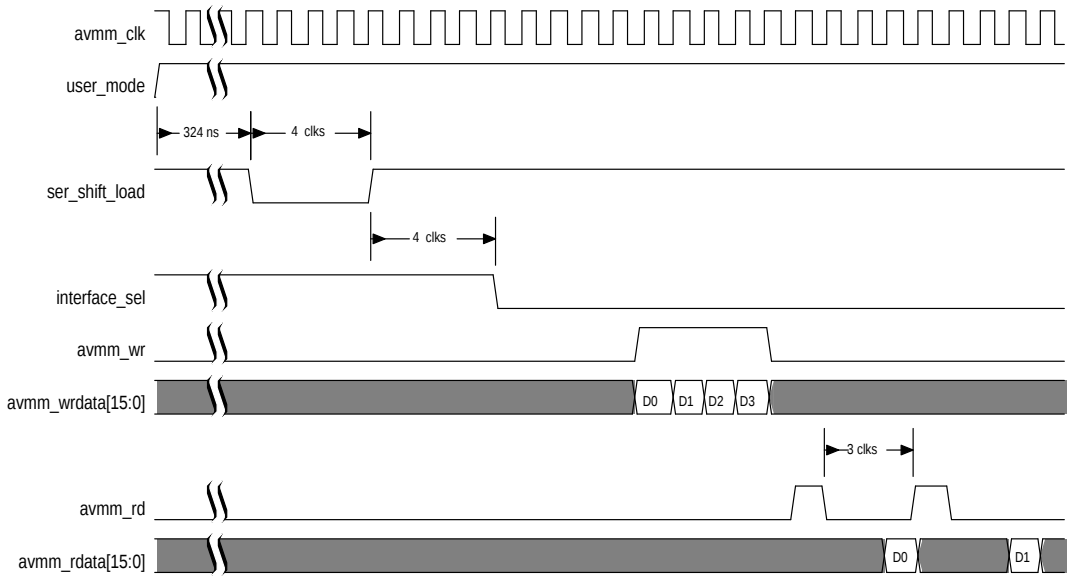
The Hard IP reconfiguration interface is consists of an Avalon-MM slave interface with a 10-bit address and 16-bit data. You can use this bus dynamically modify the value of configuration registers that are read-only at run time. To ensure proper system operation, Altera recommends that you reset or repeat device enumeration of the PCI Express link after changing the value of read-only configuration registers of the Hard IP. For a description of the registers available via this interface refer to *Hard IP Reconfiguration and Transceiver Reconfiguration*.

Table 6-7: Hard IP Reconfiguration Signals

Signal	I/O	Description
hip_reconfig_clk	I	Reconfiguration clock. The frequency range for this clock is 50–125 MHz.
hip_reconfig_rst_n	I	Active-low Avalon-MM reset. Resets all of the dynamic reconfiguration registers to their default values as described in *Hard IP Reconfiguration Registers*.
hip_reconfig_address[9:0]	I	The 10-bit reconfiguration address.
hip_reconfig_read	I	Read signal. This interface is not pipelined. You must wait for the return of the hip_reconfig_readdata[15:0] from the current read before starting another read operation.
hip_reconfig_readdata[15:0]	O	16-bit read data. hip_reconfig_readdata[15:0] is valid on the third cycle after the assertion of hip_reconfig_read.
hip_reconfig_write	I	Write signal.
hip_reconfig_writedata[15:0]	I	16-bit write model.
hip_reconfig_byte_en[1:0]	I	Byte enables, currently unused.
ser_shift_load	I	You must toggle this signal once after changing to user mode before the first access to read-only registers. This signal should remain asserted for a minimum of 324 ns after switching to user mode.
interface_sel	I	A selector which must be asserted when performing dynamic reconfiguration. Drive this signal low 4 clock cycles after the release of ser_shift_load.

Figure 6-4: Hard IP Reconfiguration Bus Timing of Read-Only Registers

The following figure shows the timing of writes and reads on the Hard IP reconfiguration bus.



For a detailed description of the Avalon-MM protocol, refer to the *Avalon Memory Mapped Interfaces* chapter in the *Avalon Interface Specifications*.

Table 6-8: Reconfiguration Block Signals

Signal	I/O	Description
hip_reconfig_clk	I	Reconfiguration clock for the Hard IP implementation. This clock should not exceed 70MHz.
hip_reconfig_rst_n	I	Active-low Avalon-MM reset. Resets all of the dynamic reconfiguration registers to their default values as described in Table 17–1 on page 17–2.
hip_reconfig_address[9:0]	I	A 10-bit address.
hip_reconfig_read	I	Read signal.
hip_reconfig_readdata[15:0]	O	16-bit read data bus.
hip_reconfig_write	I	Write signal.
hip_reconfig_writedata[15:0]	I	16-bit write data bus.

Signal	I/O	Description
hip_reconfig_byte_en[1:0]	I	Byte enables.
ser_shift_load	O	A pulse on this signal duplicates the global configuration registers to a space the you can update using the Hard IP reconfiguration signals.
interface_sel	I	Chipselect.

Related Information

- **Reconfigurable Read-Only Registers in the Hard IP for PCI Express** on page 15-1
- **Avalon Interface Specifications**

Physical Layer Interface Signals

Altera provides an integrated solution with the Transaction, Data Link and Physical Layers. 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 10 devices the SERDES entity is included in the library files for PCI Express.

Serial Interface Signals

The following table describes the serial interface signals.

Table 6-9: 1-Bit Interface Signals

Signal	I/O	Description
tx_out[7:0] (1)	O	Transmit input. These signals are the serial outputs of lanes 7–0.
rx_in[7:0] (1)	I	Receive input. These signals are the serial inputs of lanes 7–0.

Note to :

1. The ×1 IP core only has lane 0. The ×2 IP core only has lanes 1–0. The ×4 IP core only has lanes 3–0.

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

Transceiver channels are arranged in groups of six. For GX devices, the lowest six channels on the left side of the device are labeled GXB_L0, the next group is GXB_L1, and so on. Channels on the right side of the device are labeled GXB_R0, GXB_R1, and so on. Be sure to connect the Hard IP for PCI Express on the left side of the device to appropriate channels on the left side of the device, as specified in the *Pin-out Files for Altera Devices*.

Related Information

Pin-out Files for Altera Devices

PIPE Interface Signals

These PIPE signals are available for Gen1, Gen2, and Gen 3 variants so that you can simulate using either the serial or the PIPE interface. Simulation is much faster using the PIPE interface because the PIPE simulation bypasses the serdes model . For Gen1 and Gen2 variants, the PIPE interface is 8 bits. For Gen3 variants, the PIPE interface is 32 bits. You can use the PIPE interface for simulation even though your actual design includes a serial interface to the internal transceivers. However, it is not possible to use the Hard IP PIPE interface in hardware, including probing these signals using SignalTap® II Embedded Logic Analyzer.

Note: The Root Port BFM bypasses Gen3 Phase 2 and Phase 3 Equalization. You must adjust your third-party Root Port BFM to terminate Equalization after Phase 0 and Phase 1 complete.

In the following table , 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 6-10: PIPE Interface Signals

Signal	I/O	Description
`txdata0[7:0]`	O	Transmit data <*n*> (2 symbols on lane <*n*>). This bus transmits data on lane <*n*>.
`txdatak0`	O	Transmit data control <*n*>. This signal serves as the control bit for `txdata` <*n*>.
`txdatavalid0`	O	When asserted, the `txdata0[7:0]` is valid.
`rxdata0[7:0]` (2)	I	Receive data <*n*> (2 symbols on lane <*n*>). This bus receives data on lane <*n*>.
`rxdatak0` (2)	I	Receive data >*n*>. This bus receives data on lane <*n*>.
`txdetectrx0`	O	Transmit detect receive <*n*>. This signal tells the PHY layer to start a receive detection operation or to begin loopback.
`txelecidle`	O	Transmit electrical idle <*n*>. This signal forces the TX output to electrical idle.
`txcompl0`	O	Transmit compliance <*n*>. This signal forces the running disparity to negative in compliance mode (negative COM character).
`rxpolarity0`	O	Receive polarity <*n*>. This signal instructs the PHY layer to invert the polarity of the 8B/10B receiver decoding block.
`powerdown0[1:0]`	O	Power down <*n*>. This signal requests the PHY to change its power state to the specified state (P0, P0s, P1, or P2).

Signal	I/O	Description
tx_margin[2:0]	O	Transmit V_{OD} margin selection. The value for this signal is based on the value from the Link Control 2 Register. Available for simulation only.
txswing	O	When asserted, indicates full swing for the transmitter voltage. When deasserted indicates half swing.
rxvalid0 (1) (2)	I	Receive valid <*n*>. This symbol indicates symbol lock and valid data on rxdata<*n*> and rxdatak <*n*>.
phystatus0 (2)	I	PHY status <*n*>. This signal communicates completion of several PHY requests.
rxelecidle0 (2)	I	Receive electrical idle <*n*>. When asserted, indicates detection of an electrical idle.
rxs tatus0[2:0] (2)	I	Receive status <*n*>. This signal encodes receive status and error codes for the receive data stream and receiver detection.
simu_mode_pipe	I	When set to 1, the PIPE interface is in simulation mode.
sim_pipe_rate[1:0]	O	The 2-bit encodings have the following meanings: • 2'b00: Gen1 rate (2.5 Gbps) • 2'b01: Gen2 rate (5.0 Gbps) • 2'b1X: 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.
sim_pipe_pclk_out	O	TX datapath clock to the BFM PHY. pclk_out is derived from refclk and provides the source synchronous clock for TX data from the PHY.
sim_pipe_clk250_ out	O	Used to generate pclk.
sim_pipe_clk500_ out	O	Used to generate pclk.
sim_pipe_ ltssmstate0[4:0]	I and O	LTSSM state: The LTSSM state machine encoding defines the following states: • 5'b00000: Detect.Quiet • 5'b 00001: Detect.Active • 5'b00010: Polling.Active • 5'b 00011: Polling.Compliance • 5'b 00100: Polling.Configuration • 5'b00101: Polling.Speed

Signal	I/O	Description
		• 5'b00110: config.LinkwidthsStart • 5'b 00111: Config.Linkaccept • 5'b 01000: Config.Lanenumaccept • 5'b01001: Config.Lanenumwait • 5'b01010: Config.Complete • 5'b 01011: Config.Idle • 5'b01100: Recovery.Rcvlock • 5'b01101: Recovery.Rcvconfig • 5'b01110: Recovery.Idle • 5'b 01111: L0 • 5'b10000: Disable • 5'b10001: Loopback.Entry • 5'b10010: Loopback.Active • 5'b10011: Loopback.Exit • 5'b10100: Hot.Reset • 5'b10101: LOs • 5'b11001: L2.transmit.Wake • 5'b11010: Speed.Recovery • 5'b11011: Recovery.Equalization, Phase 0 • 5'b11100: Recovery.Equalization, Phase 1 • 5'b11101: Recovery.Equalization, Phase 2 • 5'b11110: Recovery.Equalization, Phase 3 • 5'b11111: Recovery.Equalization, Done
`rxfreqlocked0` (1) (2)	I	When asserted indicates that the `pclk_in` used for PIPE simulation is valid.
`eidlein-fersel0[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
`tx_deemph0`	O	Transmit de-emphasis selection. The Arria 10 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.

Signal	I/O	Description
testin_zero	O	When asserted, indicates accelerated initialization for simulation is active.

Notes:

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

Test Signals

The test_in buses provides run-time control and monitoring of the internal state of the Arria 10 Hard IP for PCI Express.

- Altera recommends that you use the test_out 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.

Table 6-11: Decoding of test_in[11:8]

test_in[11:8] Value	Signal Group
4'b0011	PIPE Interface Signals
All other values	Reserved

The following table describes the test_in bus signals. In Qsys these signals have the prefix, *hip_ctrl_*.

Table 6-12: Test Interface Signals(1), (2)

Signal	I/O	Description
test_in[31:0]	I	The bits of the test_in bus have the following definitions: • [0]: Simulation mode. This signal can be set to 1 to accelerate initialization by reducing the value of many initialization counters. • [4:1]: Reserved. Must be set to 4'b0100. • [5]: Compliance test mode. Disable/force compliance mode. 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, Gen2 and Gen3 compliance patterns. • [31:6]–Reserved. Must be set to 26'h2.

Notes::

1. All signals are per lane.
2. Refer to *PIPE Interface Signals* for definitions of the PIPE interface signals.

Related Information

PIPE Interface Signals on page 6-13

Register Descriptions 7

December 2013

UG-01145_avmm

Subscribe Send Feedback

Correspondence between Configuration Space Registers and the PCIe Specification

The following table provides a comprehensive correspondence between the Configuration Space Capability Structures and their descriptions in the *PCI Express Base Specification 2.1 and 3.0*.

Table 7-1: Correspondence Configuration Space Capability Structures and PCIe Base Specification Description

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
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
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

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
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
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	—
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

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
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
0x038	Expansion ROM Base Address	Type 1 Configuration Space Header
0x03C	Bridge Control Interrupt Pin Interrupt Line	Bridge Control Register (Offset 3Eh)
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
0x068	Message Control Next Cap Ptr Capability ID	MSI and MSI-X Capability Structures
0x06C	MSI-X Table Offset BIR	MSI and MSI-X Capability Structures
0x070	Pending Bit Array (PBA) Offset BIR	MSI and MSI-X Capability Structures

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
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
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

Related Information

PCI Express Base Specification 3.0

Configuration Space Register Content

Table 7-2: PCI Configuration Space - PCI Compatible Configuration Space Address Map

Byte Offset	Register Set
0x000:0x03C	PCI Type 0 Compatible Configuration Space Header
0x000:0x03C	PCI Type 1 Compatible Configuration Space Header
0x040:0x04C	Reserved
0x050:0x05C	MSI Capability Structure
0x060:0x064	Reserved

Byte Offset	Register Set
0x068:0x070	MSI-X Capability Structure
0x070:0x074	Reserved
0x078:0x07C	Power Management Capability Structure
0x080:0x0BC	PCI Express Capability Structure
0x0C0:0x0C4	Reserved

Table 7-3: PCI Express Extended Configuration Space

Byte Offset	Register Set
0x100:0x16C	Virtual Channel Capability Structure
0x170:0x1FC	Reserved
0x200:0x240	Vendor Specific Extended Capability Structure
0x300:0x318	Secondary PCI Express Extended Capability Structure (for Gen3 operation)
0x31C:0x7FC	Reserved
0x800:0x834	Advanced Error Reporting (AER))
0x838:0x8FF	Reserved

For comprehensive information about these registers, refer to Chapter 7 of the *PCI Express Base Specification Revision 3.0*.

Related Information

PCI Express Base Specification Revision 3.0.

Type 0 Configuration Space Registers

Endpoints store configuration data in the Type 0 Configuration Space.

Byte Offset	31:24	23:16	15:8	7:0
0x0000	Device ID		Vendor ID	
0x004	Status		Command	
0x008	Class code			Revision ID
0x00C	0x00	Header Type	0x00	Cache Line Size
0x010	BAR Registers			
0x014	BAR Registers			
0x018	BAR Registers			
0x01C	BAR Registers			
0x020	BAR Registers			
0x024	BAR Registers			
0x028	Reserved			
0x02C	Subsystem Device ID		Subsystem Vendor ID	
0x030	Expansion ROM Base Address			
0x034	Reserved			Capabilities Pointer
0x038	Reserved			
0x03C	0x00		Interrupt Pin	Interrupt Line

Type 1 Configuration Space Registers

Rootports store configuration information in the Type 1 Configuration Space.

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	BAR Registers			
0x014	BAR Registers			
0x018	Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number
0x01C	Secondary Status		I/O Limit	I/O Base
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

MSI Capability Structure

Byte Offset (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	

MSI-X Capability Structure

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				MSI-X Table BAR Indicator
0x070	MSI-X Pending Bit Array (PBA) Offset				MSI-X Pending Bit Array – BAR Indicator

Power Management Capability Structure

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	

PCI Express AER Extended Capability Structure

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	

PCI Express Capability Structure

In the following table showing the PCI Express Capability Structure, registers that are not applicable to a device are reserved.

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	
0x08C	Link Capabilities		
0x090	Link Status	Link Control	
0x094	Slot Capabilities		
0x098	Slot Status	Slot Control	
0x09C	Root Capabilities	Root Control	
0x0A0	Root Status		
0x0A4	Device Capabilities 2		
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	

Altera-Defined Vendor Specific Extended Capability (VSEC)

The Altera-Defined Vendor Specific Extended Capability. This extended capability structure supports Configuration via Protocol (CvP) programming and detailed internal error reporting.

Byte Offset	Register Name			
	31:20	19:16	15:8	7:0
0x200	Next Capability Offset	Version	Altera-Defined VSEC Capability Header	
0x204	VSEC Length	VSEC Rev	VSEC ID Altera-Defined Vendor Specific Header	
0x208	Altera Marker			
0x20C	JTAG Silicon ID DW0 JTAG Silicon ID			
0x210	JTAG Silicon ID DW1 JTAG Silicon ID			
0x214	JTAG Silicon ID DW2 JTAG Silicon ID			
0x218	JTAG Silicon ID DW3 JTAG Silicon ID			
0x21C	CvP Status		User Device or Board Type ID	
0x220	CvP Mode Control			
0x224	CvP Data2 Register			
0x228	CvP Data Register			
0x22C	CvP Programming Control Register			
0x230	Reserved			
0x234	Uncorrectable Internal Error Status Register			
0x238	Uncorrectable Internal Error Mask Register			
0x23C	Correctable Internal Error Status Register			
0x240	Correctable Internal Error Mask Register			

Altera-Defined VSEC Capability Register

Bits	Register Description	Value	Access
[15:0]	`PCI Express Extended Capability ID`. PCIe specification defined value for VSEC Capability ID.	0x000B	RO
[19:16]	`Version`. PCIe specification defined value for VSEC version.	0x1	RO
[31:20]	`Next Capability Offset`. Starting address of the next Capability Structure implemented, if any.	Variable	RO

Altera-Defined VSEC Header Register

The following table defines the fields of the `Altera Defined Vendor Specific` register. You can specify these fields when you instantiate the Hard IP. These registers are read-only at run-time.

Table 7-4: Altera-Defined Vendor Specific Header

Bits	Register Description	Value	Access
[15:0]	`VSEC ID`. A user configurable VSEC ID.	User entered	RO
[19:16]	`VSEC Revision`. A user configurable VSEC revision.	Variable	RO
[31:20]	`VSEC Length`. Total length of this structure in bytes.	0x044	RO

Altera Marker Register

Bits	Register Description	Value	Access
[31:0]	`Altera Marker`. This read only register is an additional marker. If you use the standard Altera Programmer software to configure the device with CvP, this marker provides a value that the programming software reads to ensure that it is operating with the correct VSEC.	A Device Value	RO

JTAG Silicon ID Register

Bits	Register Description	Value	Access
[127:96]	`JTAG Silicon ID DW3`	TBD	RO
[95:64]	`JTAG Silicon ID DW2`	TBD	RO
[63:32]	`JTAG Silicon ID DW1`	TBD	RO
[31:0]	`JTAG Silicon ID DW0` - This is the JTAG Silicon ID that CvP programming software reads to determine to that the correct SRAM object file (**.sof**) is being used.	TBD	RO

User Device or Board Type ID Register

Bits	Register Description	Value	Access
[15:0]	Configurable device or board type ID to specify to CvP the correct **.sof**.	Variable	RO

CvP Status Register

The `CvP Status` register allows software to monitor the CvP status signals.

Table 7-5: CvP Status

Bits	Register Description	Reset Value	Access
[15:10]	Reserved.	0x00	RO
[9]	PLD_CORE_READY. From FPGA fabric. This status bit is provided for debug.	Variable	RO
[8]	PLD_CLK_IN_USE. From clock switch module to fabric. This status bit is provided for debug.	Variable	RO
[7]	CVP_CONFIG_DONE. Indicates that the FPGA control block has completed the device configuration via CvP and there were no errors.	Variable	RO
[6]	CVP_HF_RATE_SEL. Indicates if the FPGA control block interface to the Arria 10 hard IP for PCI Express is operating half the normal frequency–62.5MHz, instead of full rate of 125MHz	Variable	RO
[5]	USERMODE. Indicates if the configurable FPGA fabric is in user mode.	Variable	RO
[4]	CVP_EN. Indicates if the FPGA control block has enabled CvP mode.	Variable	RO
[3]	CVP_CONFIG_ERROR. Reflects the value of this signal from the FPGA control block, checked by software to determine if there was an error during configuration	Variable	RO
[2]	CVP_CONFIG_READY – reflects the value of this signal from the FPGA control block, checked by software during programming algorithm	Variable	RO
[1]	Reserved.	—	—
[0]	Reserved.	—	—

CvP Mode Control Register

The CvP Mode Control register provides global control of the CvP operation.

Table 7-6: CvP Mode Control

Bits	Register Description	Reset Value	Access
[31:16]	Reserved.	0x0000	RO

Bits	Register Description	Reset Value	Access
[15:8]	CVP_NUMCLKS. Specifies the number of CvP clock cycles required for every CvP data register write. Valid values are 0x00–0x3F, where 0x00 corresponds to 64 cycles, and 0x01-0x3F corresponds to 1 to 63 clock cycles. The upper bits are not used, but are included in this field because they belong to the same byte enable.	0x00	RW
[7:4]	Reserved.	0x0	RO
[2]	CVP_FULLCONFIG. Request that the FPGA control block reconfigure the entire FPGA including the Arria 10 Hard IP for PCI Express, bring the PCIe link down.	1'b0	RW
[1]	HIP_CLK_SEL. Selects between PMA and fabric clock when USER_MODE = 1 and PLD_CORE_READY = 1. The following encodings are defined: • 1: Selects internal clock from PMA which is required for CVP_MODE • 0: Selects the clock from soft logic fabric. This setting should only be used when the fabric is configured in USER_MODE with a configuration file that connects the correct clock. To ensure that there is no clock switching during CvP, you should only change this value when the Hard IP for PCI Express has been idle for 10 µs and wait 10 µs after changing this value before resuming activity.	1'b0	RW
[0]	CVP_MODE. Controls whether the HIP is in CVP_MODE or normal mode. The following encodings are defined: • 1: CVP_MODE is active. Signals to the FPGA control block active and all TLPs are routed to the Configuration Space. This CVP_MODE cannot be enabled if CVP_EN = 0. • 0: The IP core is in normal mode and TLPs are route to the FPGA fabric.	1'b0	RW

Related Information

Configuration via Protocol (CvP) Implementation in Altera FPGAs User Guide

CvP Data and Data2 Registers

The following table defines the CvP Data registers. For 64-bit data, the optional CvP Data2 stores the upper 32 bits of data. Programming software should write the configuration data to these registers. If you Every write to these register sets the data output to the FPGA control block and generates <*n*> clock cycles to the FPGA control block as specified by the CVP_NUM_CLKS field in the CvP Mode Control register.

Software must ensure that all bytes in the memory write dword are enabled. You can access this register using configuration writes, alternatively, when in CvP mode, these registers can also be written by a memory write to any address defined by a memory space BAR for this device. Using memory writes should allow for higher throughput than configuration writes.

Table 7-7: CvP Data Register

Bits	Register Description	Reset Value	Access
[31:0]	Upper 32 bits of configuration data to be transferred to the FPGA control block to configure the device. You can choose 32- or 64-bit data.	0x00000000	RW
[31:0]	Lower 32 bits of configuration data to be transferred to the FPGA control block to configure the device.	0x00000000	RW

CvP Programming Control Register

This register is written by the programming software to control CvP programming.

Table 7-8: CvP Programming Control Register

Bits	Register Description	Reset Value	Access
[31:2]	Reserved.	0x0000	RO
[1]	START_XFER. Sets the CvP output to the FPGA control block indicating the start of a transfer.	1'b0	RW
[0]	CVP_CONFIG. When asserted, instructs that the FPGA control block begin a transfer via CvP.	1'b0	RW

Related Information

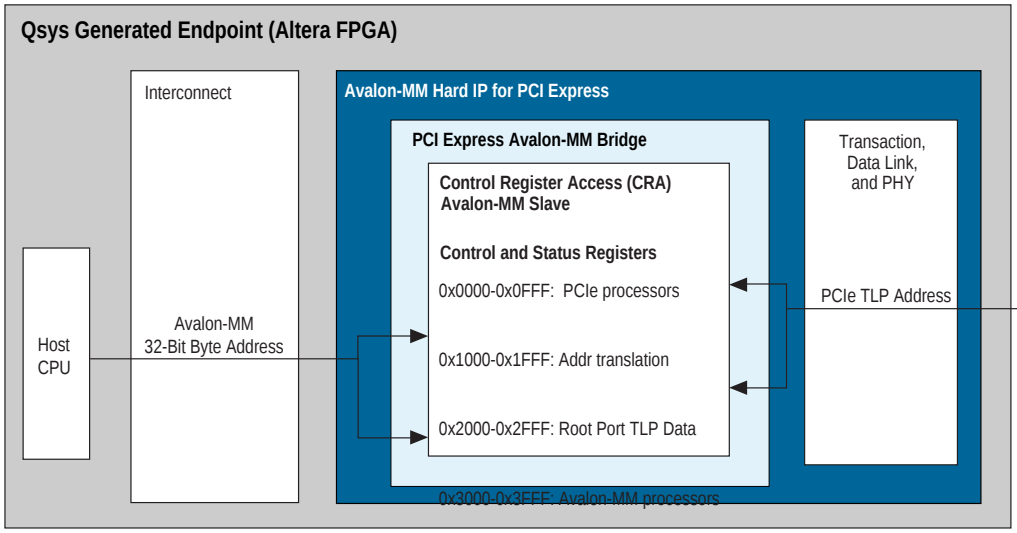
Configuration via Protocol (CvP) Implementation in Altera FPGAs User Guide

64-, 128-, or 256-Bit Avalon-MM Bridge Register Descriptions

Control and status registers in the PCI Express Avalon-MM bridge are implemented in the CRA slave module. The control registers are accessible through the Avalon-MM slave port of the CRA slave module. This module is optional. However, you must include it to access the registers.

The control and status register address space is 16 KBytes. Each 4-KByte sub-region contains a set of functions, which may be specific to accesses from the PCI Express Root Complex only, from Avalon-MM processors only, or from both types of processors. Because all accesses come across the interconnect fabric—requests from the Avalon-MM Arria 10 Hard IP for PCI Express are routed through the interconnect fabric—hardware does not enforce restrictions to limit individual processor access to specific regions. However, the regions are designed to enable straight-forward enforcement by processor software. The following figure illustrates accesses to the Avalon-MM control and status registers from the Host CPU and PCI Express link.

Figure 7-1: Accesses to the Avalon-MM Bridge Control and Status Register



The following table describes the four subregions.

Table 7-9: Avalon-MM Control and Status Register Address Spaces

AddressRange	Address Space Usage
0x0000-0x0FFF	Registers typically intended for access by PCI Express processors only. This includes PCI Express interrupt enable controls, write access to the PCI Express Avalon-MM bridge mailbox registers, and read access to Avalon-MM-to-PCI Express mailbox registers.
0x1000-0x1FFF	Avalon-MM-to-PCI Express address translation tables. Depending on the system design these may be accessed by PCI Express processors, Avalon-MM processors, or both.
0x2000-0x2FFF	Root Port request registers. An embedded processor, such as the Nios II processor, programs these registers to send the data to send Configuration TLPs, I/O TLPs, single dword Memory Reads and Write request, and receive interrupts from an Endpoint.
0x3000-0x3FFF	Registers typically intended for access by Avalon-MM processors only. These include Avalon-MM interrupt enable controls, write access to the Avalon-MM-to-PCI Express mailbox registers, and read access to PCI Express Avalon-MM bridge mailbox registers.

Note: The data returned for a read issued to any undefined address in this range is unpredictable.

The following table lists the complete address map for the PCI Express Avalon-MM bridge registers.

Note: In the following table the text in green are links to the detailed register description

Table 7-10: PCI Express Avalon-MM Bridge Register Map

Address Range	Register
0x0040	Avalon-MM to PCI Express Interrupt Status Register
0x0050	Avalon-MM to PCI Express Interrupt Status Enable Register
0x0060	Avalon-MM Interrupt Vector Register
0x0800–0x081F	PCI Express-to-Avalon-MM Mailbox Registers
0x0900–x091F	Avalon-MM to PCI Express Mailbox Registers
0x1000–0x1FFF	Avalon-MM to PCI Express Address Translation Table
0x2000–0x2FFF	Root Port TLP Data Registers
0x3060	Avalon-MM to PCI Express Interrupt Status Registers for Root Ports
0x3060	PCI Express to Avalon-MM Interrupt Status Register for Endpoints
0x3070	INT-X Interrupt Enable Register for Root Ports
0x3070	INT-X Interrupt Enable Register for Endpoints
0x3B00-0x3B1F	PCI Express to Avalon-MM Mailbox Registers
0x3A00-0x3A1F	Avalon-MM to PCI Express Mailbox Registers

Avalon-MM to PCI Express Interrupt Registers

Avalon-MM to PCI Express Interrupt Status Registers

These registers contain status of various signals in the PCI Express Avalon-MM bridge logic and allow PCI Express interrupts to be asserted when enabled. Only Root Complexes should access these registers; however, hardware does not prevent other Avalon-MM masters from accessing them.

Table 7-11: Avalon-MM to PCI Express Interrupt Status Register0x0040

Bit	Name	Access	Description
[31:24]	Reserved	—	—
[23]	A2P_MAILBOX_INT7	RW1C	1 when the A2P_MAILBOX7 is written to
[22]	A2P_MAILBOX_INT6	RW1C	1 when the A2P_MAILBOX6 is written to
[21]	A2P_MAILBOX_INT5	RW1C	1 when the A2P_MAILBOX5 is written to

Bit	Name	Access	Description
[20]	A2P_MAILBOX_INT4	RW1C	1 when the A2P_MAILBOX4 is written to
[19]	A2P_MAILBOX_INT3	RW1C	1 when the A2P_MAILBOX3 is written to
[18]	A2P_MAILBOX_INT2	RW1C	1 when the A2P_MAILBOX2 is written to
[17]	A2P_MAILBOX_INT1	RW1C	1 when the A2P_MAILBOX1 is written to
[16]	A2P_MAILBOX_INT0	RW1C	1 when the A2P_MAILBOX0 is written to
[15:0]	AVL_IRQ_ASSERTED[15:0]	RO	Current value of the Avalon-MM interrupt (IRQ) input ports to the Avalon-MM RX master port: • 0 – Avalon-MM IRQ is not being signaled. • 1 – Avalon-MM IRQ is being signaled. A Qsys-generated IP Compiler for PCI Express has as many as 16 distinct IRQ input ports. Each AVL_IRQ_ASSERTED[] bit reflects the value on the corresponding IRQ input port.

Avalon-MM to PCI Express Interrupt Enable Registers

A PCI Express interrupt can be asserted for any of the conditions registered in the Avalon MM to PCI Express Interrupt Status register by setting the corresponding bits in the Avalon-MM-to-PCI Express Interrupt Enable register. Either MSI or legacy interrupts can be generated as explained in the section "Enabling MSI or Legacy Interrupts" on page 13–7. (Note to reformatter: Chapter 13 - Interrupts.)

Table 7-12: Avalon-MM to PCI Express Interrupt Enable Register0x0050

Bits	Name	Access	Description
[31:24]	Reserved	—	—
[23:16]	A2P_MB_IRQ	RW	Enables generation of PCI Express interrupts when a specified mailbox is written to by an external Avalon-MM master.

Bits	Name	Access	Description
[15:0]	AVL_IRQ[15:0]	RX	Enables generation of PCI Express interrupts when a specified Avalon-MM interrupt signal is asserted. Your Qsys system may have as many as 16 individual input interrupt signals.

Avalon-MM to PCI Express Interrupt Vector Register

Table 7-13: Avalon-MM Interrupt Vector Register0x0060

Bits	Name	Access	Description
[31:5]	Reserved	—	—
[4:0]	AVALON_IRQ_VECTOR	RO	Stores the interrupt vector of the system interconnect fabric. The host software should read this register after being interrupted and determine the servicing priority.

PCI Express Mailbox Registers

The PCI Express Root Complex typically requires write access to a set of PCI Express-to-Avalon-MM mailbox registers and read-only access to a set of Avalon-MM-to-PCI Express mailbox registers. Eight mailbox registers are available.

The PCI Express-to-Avalon-MM Mailbox registers are writable at the addresses shown in the following table. Writing to one of these registers causes the corresponding bit in the Avalon-MM Interrupt Status register to be set to a one.

Table 7-14: PCI Express-to-Avalon-MM Mailbox Registers0x0800–0x081F

Address	Name	Access	Description
0x0800	P2A_MAILBOX0	RW	PCI Express-to-Avalon-MM Mailbox 0
0x0804	P2A_MAILBOX1	RW	PCI Express-to-Avalon-MM Mailbox 1
0x0808	P2A_MAILBOX2	RW	PCI Express-to-Avalon-MM Mailbox 2
0x080C	P2A_MAILBOX3	RW	PCI Express-to-Avalon-MM Mailbox 3
0x0810	P2A_MAILBOX4	RW	PCI Express-to-Avalon-MM Mailbox 4
0x0814	P2A_MAILBOX5	RW	PCI Express-to-Avalon-MM Mailbox 5
0x0818	P2A_MAILBOX6	RW	PCI Express-to-Avalon-MM Mailbox 6
0x081C	P2A_MAILBOX7	RW	PCI Express-to-Avalon-MM Mailbox 7

The Avalon-MM-to-PCI Express Mailbox registers are read at the addresses shown in the following table. The PCI Express Root Complex should use these addresses to read the mailbox information after being signaled by the corresponding bits in the Avalon MM to PCI Express Interrupt Status register.

Table 7-15: Avalon-MM-to-PCI Express Mailbox Registers0x0900–0x091F

Address	Name	Access	Description
0x0900	A2P_MAILBOX0	RO	Avalon-MM-to-PCI Express Mailbox 0
0x0904	A2P_MAILBOX1	RO	Avalon-MM-to-PCI Express Mailbox 1
0x0908	A2P_MAILBOX2	RO	Avalon-MM-to-PCI Express Mailbox 2
0x090C	A2P_MAILBOX3	RO	Avalon-MM-to-PCI Express Mailbox 3
0x0910	A2P_MAILBOX4	RO	Avalon-MM-to-PCI Express Mailbox 4
0x0914	A2P_MAILBOX5	RO	Avalon-MM-to-PCI Express Mailbox 5
0x0918	A2P_MAILBOX6	RO	Avalon-MM-to-PCI Express Mailbox 6
0x091C	A2P_MAILBOX7	RO	Avalon-MM-to-PCI Express Mailbox 7

Avalon-MM-to-PCI Express Address Translation Table

The Avalon-MM-to-PCI Express address translation table is writable using the CRA slave port. Each entry in the PCI Express address translation table is 8 bytes wide, regardless of the value in the current PCI Express address width parameter. Therefore, register addresses are always the same width, regardless of PCI Express address width.

These table entries are repeated for each address specified in the **Number of address pages** parameter. If **Number of address pages** is set to the maximum of 512, 0x1FF8 contains A2P_ADDR_MAP_LO511 and 0x1FFC contains A2P_ADDR_MAP_HI511.

Table 7-16: Avalon-MM-to-PCI Express Address Translation Table 0x1000–0x1FFF

Address	Bits	Name	Access	Description
0x1000	[1:0]	A2P_ADDR_SPACE0	RW	Address space indication for entry 0. Refer to Table 9–31 for the definition of these bits.
	[31:2]	A2P_ADDR_MAP_LO0	RW	Lower bits of Avalon-MM-to-PCI Express address map entry 0.
0x1004	[31:0]	A2P_ADDR_MAP_HI0	RW	Upper bits of Avalon-MM-to-PCI Express address map entry 0.

Address	Bits	Name	Access	Description
0x1008	[1:0]	A2P_ADDR_SPACE1	RW	Address space indication for entry 1. Refer to the following encodings are defined: • 2'b00:. Memory Space, 32-bit PCI Express address. 32-bit header is generated. Address bits 63:32 of the translation table entries are ignored. • 2'b01: Memory space, 64-bit PCI Express address. 64-bit address header is generated. • 2'b10: Reserved • 2'b11: Reserved
	[31:2]	A2P_ADDR_MAP_LO1	RW	Lower bits of Avalon-MM-to-PCI Express address map entry 1. This entry is only implemented if the number of address translation table entries is greater than 1.
0x100C	[31:0]	A2P_ADDR_MAP_HI1	RW	Upper bits of Avalon-MM-to-PCI Express address map entry 1. This entry is only implemented if the number of address translation table entries is greater than 1.

PCI Express to Avalon-MM Interrupt Status and Enable Registers for Endpoints

The registers in this section contain status of various signals in the PCI Express Avalon-MM bridge logic and allow Avalon interrupts to be asserted when enabled. A processor local to the interconnect fabric that processes the Avalon-MM interrupts can access these registers.

Note: These registers must not be accessed by the PCI Express Avalon-MM bridge master ports; however, there is nothing in the hardware that prevents a PCI Express Avalon-MM bridge master port from accessing these registers.

The following table describes the Interrupt Status register when you configure the core as an Endpoint. It records the status of all conditions that can cause an Avalon-MM interrupt to be asserted.

Table 7-17: PCI Express to Avalon-MM Interrupt Status Register for Endpoints 0x3060

Bits	Name	Access	Description
0	ERR_PCI_WRITE_FAILURE	RW1C	When set to 1, indicates a PCI Express write failure. This bit can also be cleared by writing a 1 to the same bit in the Avalon MM to PCI Express Interrupt Status register.

Bits	Name	Access	Description
1	ERR_PCI_READ_FAILURE	RW1C	When set to 1, indicates the failure of a PCI Express read. This bit can also be cleared by writing a 1 to the same bit in the Avalon MM to PCI Express Interrupt Status register.
[15:2]	Reserved	—	—
[16]	P2A_MAILBOX_INT0	RW1C	1 when the P2A_MAILBOX0 is written
[17]	P2A_MAILBOX_INT1	RW1C	1 when the P2A_MAILBOX1 is written
[18]	P2A_MAILBOX_INT2	RW1C	1 when the P2A_MAILBOX2 is written
[19]	P2A_MAILBOX_INT3	RW1C	1 when the P2A_MAILBOX3 is written
[20]	P2A_MAILBOX_INT4	RW1C	1 when the P2A_MAILBOX4 is written
[21]	P2A_MAILBOX_INT5	RW1C	1 when the P2A_MAILBOX5 is written
[22]	P2A_MAILBOX_INT6	RW1C	1 when the P2A_MAILBOX6 is written
[23]	P2A_MAILBOX_INT7	RW1C	1 when the P2A_MAILBOX7 is written
[31:24]	Reserved	—	—

An Avalon-MM interrupt can be asserted for any of the conditions noted in the Avalon-MM Interrupt Status register by setting the corresponding bits in the PCI Express to Avalon-MM Interrupt Enable register.

PCI Express interrupts can also be enabled for all of the error conditions described. However, it is likely that only one of the Avalon-MM or PCI Express interrupts can be enabled for any given bit because typically a single process in either the PCI Express or Avalon-MM domain is responsible for handling the condition reported by the interrupt.

Table 7-18: INT-X Interrupt Enable Register for Endpoints 0x3070

Bits	Name	Access	Description
[31:0]	PCI Express to Avalon-MM Interrupt Enable	RW	When set to 1, enables the interrupt for the corresponding bit in the PCI Express to Avalon MM Interrupt Status register to cause the Avalon Interrupt signal (cra_Irq_o) to be asserted. Only bits implemented in the PCI Express to Avalon MM Interrupt Status register are implemented in the Enable register. Reserved bits cannot be set to a 1.

Avalon-MM Mailbox Registers

A processor local to the interconnect fabric typically requires write access to a set of Avalon-MM-to-PCI Express Mailbox registers and read-only access to a set of PCI Express-to-Avalon-MM Mailbox registers. Eight mailbox registers are available.

The Avalon-MM-to-PCI Express Mailbox registers are writable at the addresses shown in the following table. When the Avalon-MM processor writes to one of these registers the corresponding bit in the Avalon MM to PCI Express Interrupt Status register is set to 1.

Table 7-19: Avalon-MM to PCI Express Mailbox Registers 0x3A00–0x3A1F

Address	Name	Access	Description
0x3A00	A2P_MAILBOX0	RW	Avalon-MM-to-PCI Express mailbox 0
0x3A04	A2P_MAILBOX1	RW	Avalon-MM-to-PCI Express mailbox 1
0x3A08	A2P _MAILBOX2	RW	Avalon-MM-to-PCI Express mailbox 2
0x3A0C	A2P _MAILBOX3	RW	Avalon-MM-to-PCI Express mailbox 3
0x3A10	A2P _MAILBOX4	RW	Avalon-MM-to-PCI Express mailbox 4
0x3A14	A2P _MAILBOX5	RW	Avalon-MM-to-PCI Express mailbox 5
0x3A18	A2P _MAILBOX6	RW	Avalon-MM-to-PCI Express mailbox 6

Address	Name	Access	Description
0x3A1C	A2P_MAILBOX7	RW	Avalon-MM-to-PCI Express mailbox 7

The PCI Express-to-Avalon-MM Mailbox registers are read-only at the addresses shown in the following table. The Avalon-MM processor reads these registers when the corresponding bit in the PCI Express to Avalon-MM Interrupt Status register is set to 1.

Table 7-20: PCI Express to Avalon-MM Mailbox Registers 0x3B00–0x3B1F

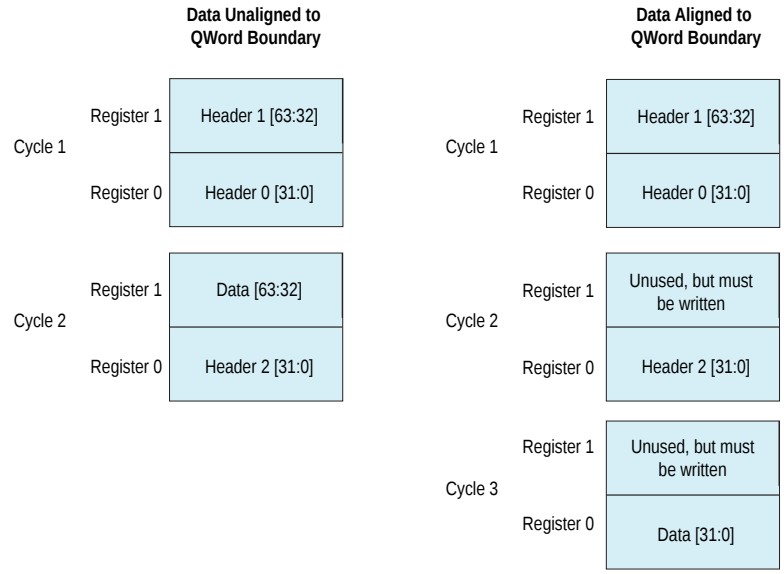
Address	Name	Access Mode	Description
0x3B00	P2A_MAILBOX0	RO	PCI Express-to-Avalon-MM mailbox 0
0x3B04	P2A_MAILBOX1	RO	PCI Express-to-Avalon-MM mailbox 1
0x3B08	P2A_MAILBOX2	RO	PCI Express-to-Avalon-MM mailbox 2
0x3B0C	P2A_MAILBOX3	RO	PCI Express-to-Avalon-MM mailbox 3
0x3B10	P2A_MAILBOX4	RO	PCI Express-to-Avalon-MM mailbox 4
0x3B14	P2A_MAILBOX5	RO	PCI Express-to-Avalon-MM mailbox 5
0x3B18	P2A_MAILBOX6	RO	PCI Express-to-Avalon-MM mailbox 6
0x3B1C	P2A_MAILBOX7	RO	PCI Express-to-Avalon-MM mailbox 7

Programming Model for Avalon-MM Root Port

The Application Layer writes the Root Port TLP TX Data registers with TLP formatted data for Configuration Read and Write Requests, Message TLPs, I/O Read and Write Requests, or single dword Memory Read and Write Requests. Software should check the Root Port Link Status register (offset 0x92) to ensure the Data Link Layer Link Active bit is set to 1'b1 before issuing a Configuration requests to downstream ports.

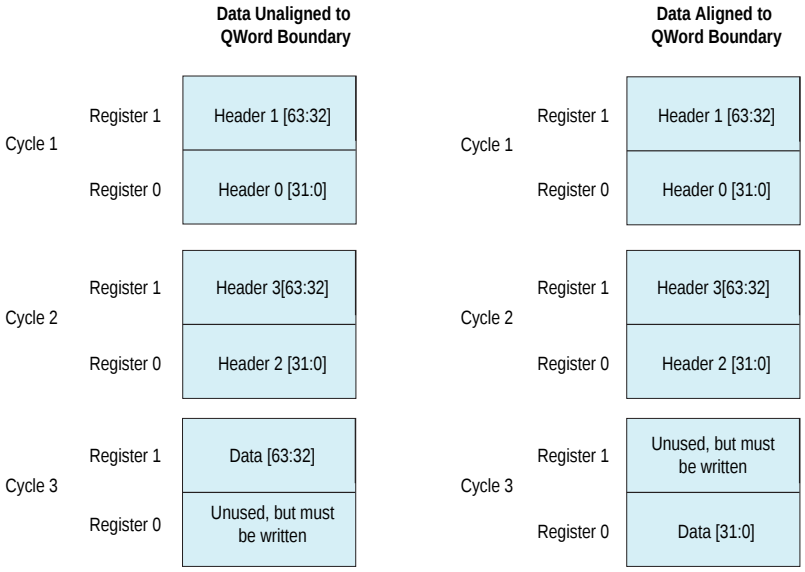
The Application Layer data must be in the appropriate TLP format with the data payload aligned to the TLP address. Aligning the payload data to the TLP address may result in the payload data being either aligned or unaligned to the qword. The following figure illustrates three dword TLPs with data that is aligned and unaligned to the qword.

Figure 7-2: Layout of Data with 3 DWord Headers



The following figure illustrates four dword TLPs with data that is aligned and unaligned to the qword.

Figure 7-3: Layout of Data with 4 DWord Headers



The TX TLP programming model scales with the data width. The Application Layer performs the same writes for both the 64- and 128-bit interfaces. The Application Layer can only have one outstanding non-posted request at a time. The Application Layer must use tags 16–31 to identify non-posted requests.

Note: For Root Ports, the Avalon-MM bridge does not filter Type 0 Configuration Requests by device number. Application Layer software should filter out all requests to Avalon-MM Root Port registers

that are not for device 0. Application Layer software should return an Unsupported Request Completion Status.

Sending a Write TLP

The Application Layer performs the following sequence of Avalon-MM accesses to the CRA slave port to send a Memory Write Request:

1. Write the first 32 bits of the TX TLP to RP_TX_REG0.
2. Write the next 32 bits of the TX TLP to RP_TX_REG1.
3. Write the RP_TX_CNTRL.SOP to 1'b1 to push the first two dwords of the TLP into the Root Port TX FIFO.
4. Repeat Steps 1 and 2. The second write to RP_TX_REG1 is required, even for three dword TLPs with aligned data.
5. If the packet is complete write RP_TX_CNTRL to 2'b10 o indicate the end of the packet. If the packet is not complete write 2'b00 to RP_TX_CNTRL.
6. Repeat this sequence to program a complete TLP.

When the programming of the TX TLP is complete, the Avalon-MM Bridge schedules the TLP with higher priority than TX TLPs coming from the TX slave port.

Receiving a Completion TLP

The Completion TLPs associated with the Non-Posted TX requests are stored in the RP_RX_CPL FIFO buffer and subsequently loaded into RP_RXCPL registers. The Application Layer performs the following sequence to retrieve the TLP.

1. Polls the RP_RXCPL_STA TUS.SOP to determine when it is set to 1'b1.
2. Then RP_RXCPL_STATUS.SOP = 1'b'1, reads RP_RXCPL_REG0 and RP_RXCPL_REG1 to retrieve dword 0 and dword 1 of the Completion TLP.
3. Read the RP_RXCPL_STATUS.EOP.

- If RP_RXCPL_STATUS.EOP = 1'b0, read RP_RXCPL_REG0 and RP_RXCPL_REG1 to retrieve dword 2 and dword 3 of the Completion TLP, then repeat step 3.
- If RP_RXCPL_STATUS.EOP = 1'b1, read RP_RXCPL_REG0 and RP_RXCPL_REG1 to retrieve final dwords of TLP.

PCI Express to Avalon-MM Interrupt Status and Enable Registers for Root Ports

The Root Port supports MSI, MSI-X and legacy (INTx) interrupts. MSI and MSI-X interrupts are memory writes from the Endpoint to the Root Port. MSI and MSI-X requests are forwarded to the interconnect without asserting CraIrq_o.

Table 7-21: Avalon-MM Interrupt Status Registers for Root Ports 0x3060

Bits	Name	Access Mode	Description
[31:5]	Reserved	—	—

Bits	Name	Access Mode	Description
[4]	RPRX_CPL_RECEIVED	RW1C	Set to 1'b1 when the Root Port has received a Completion TLP for an outstanding Non-Posted request from the TLP Direct channel.
[3]	INTD_RECEIVED	RW1C	The Root Port has received INTD from the Endpoint.
[2]	INTC_RECEIVED	RW1C	The Root Port has received INTC from the Endpoint.
[1]	INTB_RECEIVED	RW1C	The Root Port has received INTB from the Endpoint.
[0]	INTA_RECEIVED	RW1C	The Root Port has received INTA from the Endpoint.

Table 7-22: INT-X Interrupt Enable Register for Root Ports 0x3070

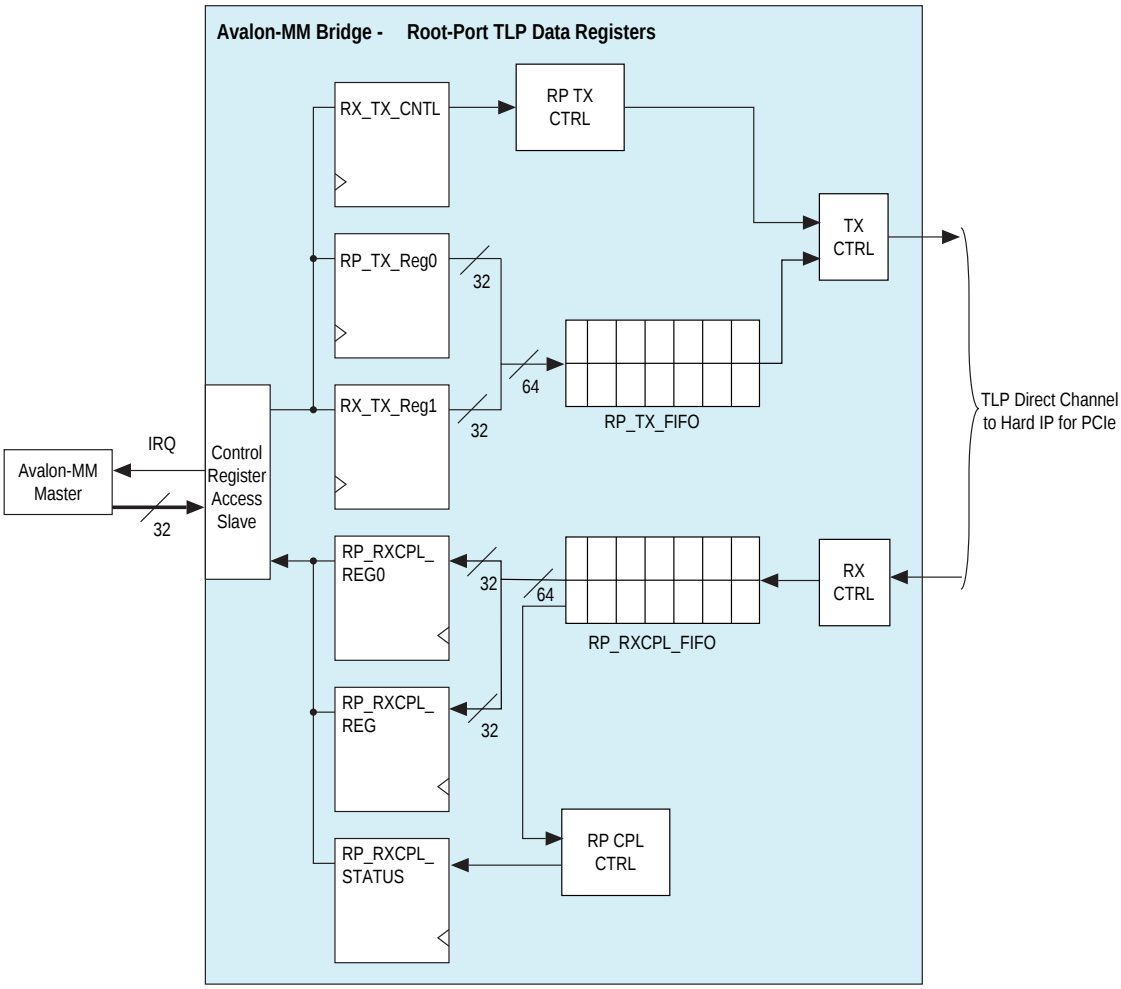
Bit	Name	Access Mode	Description
[31:5]	Reserved	—	—
[4]	RPRX_CPL_RECEIVED	RW	When set to 1'b1, enables the assertion of CraIrq_o when the Root Port Interrupt Status register RPRX_CPL_RECEIVED bit indicates it has received a Completion for a Non-Posted request from the TLP Direct channel.
[3]	INTD_RECEIVED_ENA	RW	When set to 1'b1, enables the assertion of CraIrq_o when the Root Port Interrupt Status register INTD_RECEIVED bit indicates it has received INTD.
[2]	INTC_RECEIVED_ENA	RW	When set to 1'b1, enables the assertion of CraIrq_o when the Root Port Interrupt Status register INTC_RECEIVED bit indicates it has received INTC.
[1]	INTB_RECEIVED_ENA	RW	When set to 1'b1, enables the assertion of CraIrq_o when the Root Port Interrupt Status register INTB_RECEIVED bit indicates it has received INTB.

Bit	Name	Access Mode	Description
[0]	INTA_RECEIVED_ENA	RW	When set to 1'b1, enables the assertion of CraIrq_o when the Root Port Interrupt Status register INTA_RECEIVED bit indicates it has received INTA.

Root Port TLP Data Registers

The TLP data registers provide a mechanism for the Application Layer to specify data that the Root Port uses to construct Configuration TLPs, I/O TLPs, and single dword Memory Reads and Write requests. The Root Port then drives the TLPs on the TLP Direct Channel to access the Configuration Space, I/O space, or Endpoint memory.

Figure 7-4: Root Port TLP Data Registers



Note: The high performance TLPs implemented by Avalon-MM ports in the Avalon-MM Bridge are also available for Root Ports. For more information about these TLPs, refer to Avalon-MM Bridge TLPs.

Table 7-23: Root Port TLP Data Registers 0x2000–0x2FFF

Root-Port Request Registers				Address Range: 0x2800-0x2018
Address	Bits	Name	Access	Description
0x2000	[31:0]	RP_TX_REG0	W	Lower 32 bits of the TX TLP.
0x2004	[31:0]	RP_TX_REG1	W	Upper 32 bits of the TX TLP.
0x2008	[31:2]	Reserved	—	—
	[1]	RP_TX_CNTRL.EOP	W	Write 1'b1 to specify the of end a packet. Writing this bit frees the corresponding entry in the FIFO.
	[0]	RP_TX_CNTRL.SOP	W	Write 1'b1 to specify the start of a packet.
0x2010	[31:16]	Reserved	—	—
	[15:8]	RP_RXCPL_STATUS	R	Specifies the number of words in the RX completion FIFO that contain valid data.
	[7:2]	Reserved	—	—
	[1]	RP_RXCPL_ STATUS.EOP	R	When 1'b1, indicates that the data for a Completion TLP is ready to be read by the Application Layer. The Application Layer must poll this bit to determine when a Completion TLP is available.
	[0]	RP_RXCPL_ STATUS.SOP	R	When 1'b1, indicates that the final data for a Completion TLP is ready to be read by the Application Layer. The Application Layer must poll this bit to determine when the final data for a Completion TLP is available.
0x2014	[31:0]	RP_RXCPL_REG1	RC	Lower 32 bits of a Completion TLP. Reading frees this entry in the FIFO.
0x2018	[31:0]	RP_RXCPL_REG1	RC	Upper 32 bits of a Completion TLP. Reading frees this entry in the FIFO.

Uncorrectable Internal Error Mask Register

The following table defines the `Uncorrectable Internal Error Mask` register. This register controls which errors are forwarded as internal uncorrectable errors. With the exception of the configuration

error detected in CvP mode, all of the errors are severe and may place the device or PCIe link in an inconsistent state. The configuration error detected in CvP mode may be correctable depending on the design of the programming software.

Table 7-24: Uncorrectable Internal Error Mask Register

Bits	Register Description	Reset Value	Access
[31:12]	Reserved.	1b'0	RO
[11]	Mask for RX buffer posted and completion overflow error.	1b'1	RWS
[10]	Reserved	1b'0	RO
[9]	Mask for parity error detected on Configuration Space to TX bus interface.	1b'1	RWS
[8]	Mask for parity error detected on the TX to Configuration Space bus interface.	1b'1	RWS
[7]	Mask for parity error detected at TX Transaction Layer error.	1b'1	RWS
[6]	Reserved	1b'0	RO
[5]	Mask for configuration errors detected in CvP mode.	1b'0	RWS
[4]	Mask for data parity errors detected during TX Data Link LCRC generation.	1b'1	RWS
[3]	Mask for data parity errors detected on the RX to Configuration Space Bus interface.	1b'1	RWS
[2]	Mask for data parity error detected at the input to the RX Buffer.	1b'1	RWS
[1]	Mask for the retry buffer uncorrectable ECC error.	1b'1	RWS
[0]	Mask for the RX buffer uncorrectable ECC error.	1b'1	RWS

Uncorrectable Internal Error Status Register

This register reports the status of the internally checked errors that are uncorrectable. When specific errors are enabled by the `Uncorrectable Internal Error Mask` register, they are handled as Uncorrectable Internal Errors as defined in the *PCI Express Base Specification 3.0*. This register is for debug only. It should only be used to observe behavior, not to drive logic custom logic.

Table 7-25: Uncorrectable Internal Error Status Register

Bits	Register Description	Access
[31:12]	Reserved.	RO
[11]	When set, indicates an RX buffer overflow condition in a posted request or Completion	RW1CS
[10]	Reserved.	RO
[9]	When set, indicates a parity error was detected on the Configuration Space to TX bus interface	RW1CS
[8]	When set, indicates a parity error was detected on the TX to Configuration Space bus interface	RW1CS
[7]	When set, indicates a parity error was detected in a TX TLP and the TLP is not sent.	RW1CS
[6]	When set, indicates that the Application Layer has detected an uncorrectable internal error.	RW1CS
[5]	When set, indicates a configuration error has been detected in CvP mode which is reported as uncorrectable. This bit is set whenever a `CVP_CONFIG_ERROR` rises while in `CVP_MODE`.	RW1CS
[4]	When set, indicates a parity error was detected by the TX Data Link Layer.	RW1CS
[3]	When set, indicates a parity error has been detected on the RX to Configuration Space bus interface.	RW1CS
[2]	When set, indicates a parity error was detected at input to the RX Buffer.	RW1CS
[1]	When set, indicates a retry buffer uncorrectable ECC error.	RW1CS
[0]	When set, indicates a RX buffer uncorrectable ECC error.	RW1CS

Related Information

PCI Express Base Specification 3.0

Correctable Internal Error Mask Register

The `Correctable Internal Error Mask` register controls which errors are forwarded as Internal Correctable Errors. This register is for debug only.

Table 7-26: Correctable Internal Error Mask Register

Bits	Register Description	Reset Value	Access
[31:7]	Reserved.	0	RO

Bits	Register Description	Reset Value	Access
[6]	Mask for Corrected Internal Error reported by the Application Layer.	1	RWS
[5]	Mask for configuration error detected in CvP mode.	0	RWS
[4:2]	Reserved.	0	RO
[1]	Mask for retry buffer correctable ECC error.	1	RWS
[0]	Mask for RX Buffer correctable ECC error.	1	RWS

Correctable Internal Error Status Register

The `Correctable Internal Error Status` register reports the status of the internally checked errors that are correctable. When these specific errors are enabled by the `Correctable Internal Error Mask` register, they are forwarded as Correctable Internal Errors as defined in the *PCI Express Base Specification 3.0*. This register is for debug only. It should only be used to observe behavior, not to drive logic custom logic.

Table 7-27: Correctable Internal Error Status Register

Bits	Register Description	Reset Value	Access
[31:6]	Reserved.	0	RO
[5]	When set, indicates a configuration error has been detected in CvP mode which is reported as correctable. This bit is set whenever a `CVP_CONFIG_ERROR` occurs while in `CVP_MODE`.	0	RW1CS
[4:2]	Reserved.	0	RO
[1]	When set, the retry buffer correctable ECC error status indicates an error.	0	RW1CS
[0]	When set, the RX buffer correctable ECC error status indicates an error.	0	RW1CS

Related Information

PCI Express Base Specification 3.0

Reset and Clocks 8

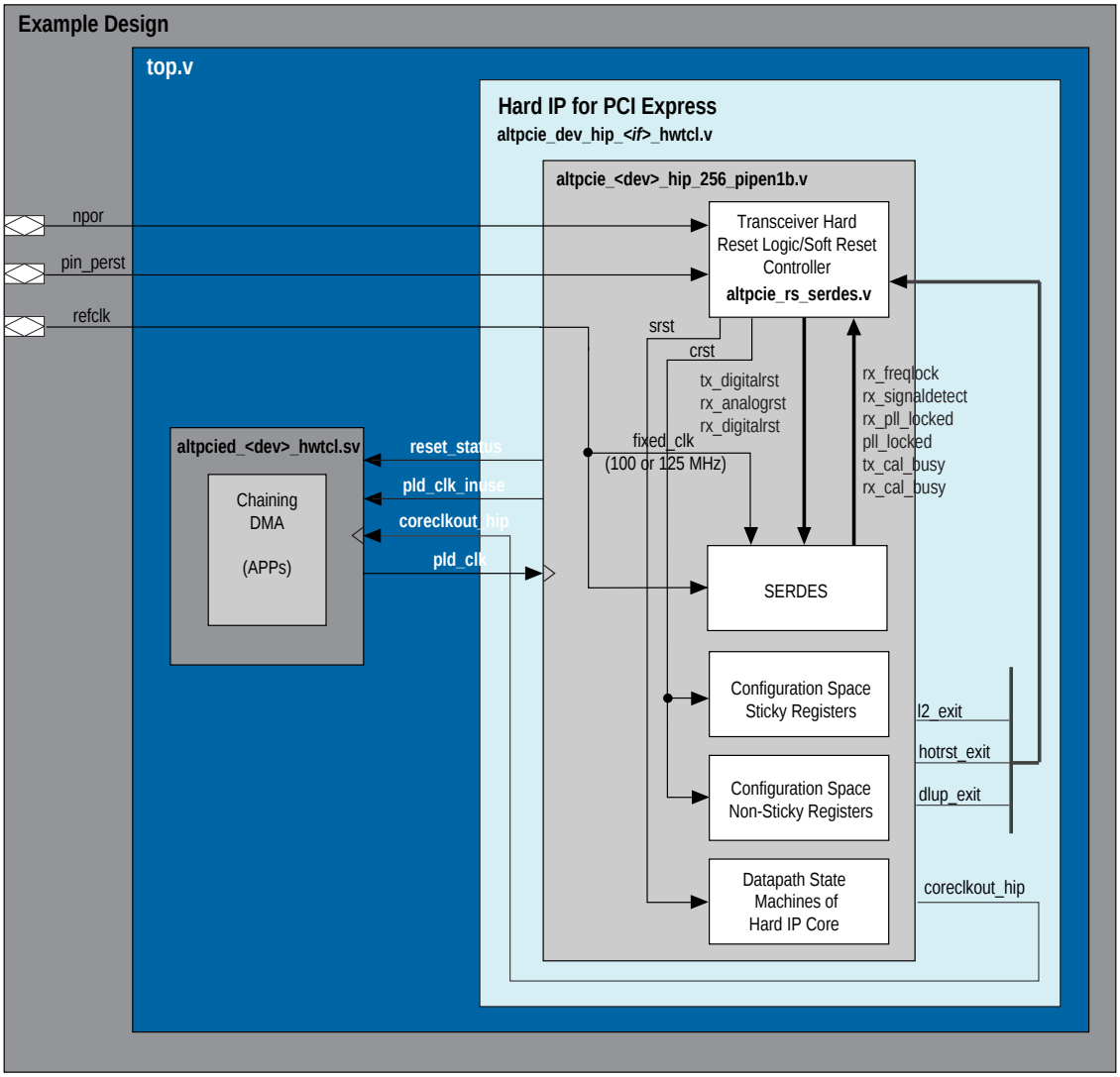
December 2013

UG-01145_avmm Subscribe Send Feedback

Reset

The following figure provides an overview of the hard reset controller included in the Arria 10 Hard IP for PCI Express IP core.

Figure 8-1: Reset Controller in Arria 10 Devices

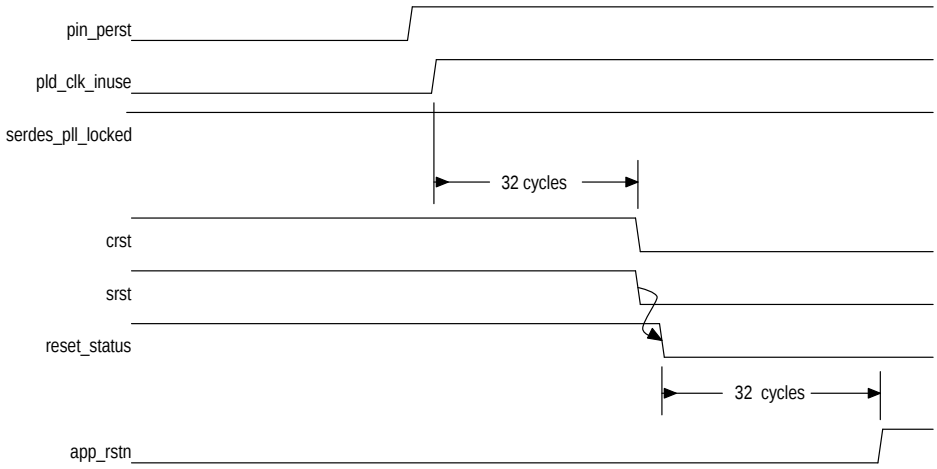


Note: Your Application Layer could instantiate a module similar to **altpcie_rs_hip.v** as shown inthe following figure to generate `app_rstn` which resets the Application Layer logic.

Reset Sequence for Hard IP for PCI Express IP Core and Application Layer

The following illustrates the reset sequence for the Hard IP for PCI Express IP core and the Application Layer logic.

Figure 8-2: Hard IP for PCI Express and Application Logic Reset Sequence



As this figure illustrates, this reset sequence includes the following steps:

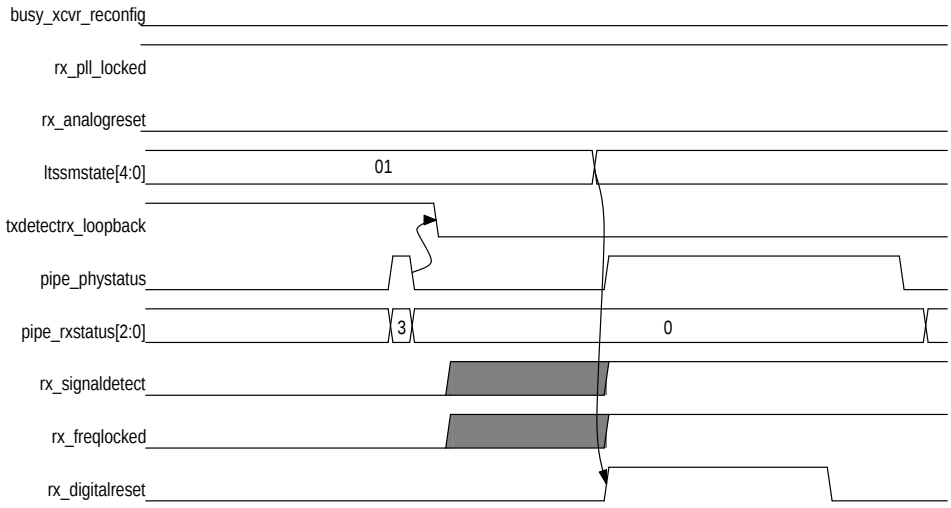
1. After `pin_perst` or npor is released, the Hard IP reset controller waits for `pld_clk_inuse` to be asserted.
2. `csrt` and `srst` are released 32 cycles after `pld_clk_inuse` is asserted.
3. The Hard IP for PCI Express deasserts the `reset_status` output to the Application Layer.
4. The **altpcied**_<*device*>**v_hwtcl.sv** deasserts `app_rstn` 32 cycles after `reset_status` is released.

Reset Sequence for RX Transceiver

The following figure illustrates the RX transceiver reset sequence. It includes the following steps:

1. After `rx_pll_locked` is asserted, the LTSSM state machine transitions from the Detect.Quiet to the Detect.Active state.
2. When the `pipe_phystatus` pulse is asserted and `pipe_rxstatus[2:0]` = 3, the receiver detect operation has completed.
3. The LTSSM state machine transitions from the Detect.Active state to the Polling.Active state.
4. The Hard IP for PCI Express asserts `rx_digitalreset`. The `rx_digitalreset signal` is deasserted after `rx_signaldetect` is stable for a minimum of 3 ms.

Figure 8-3: RX Transceiver Reset Sequence

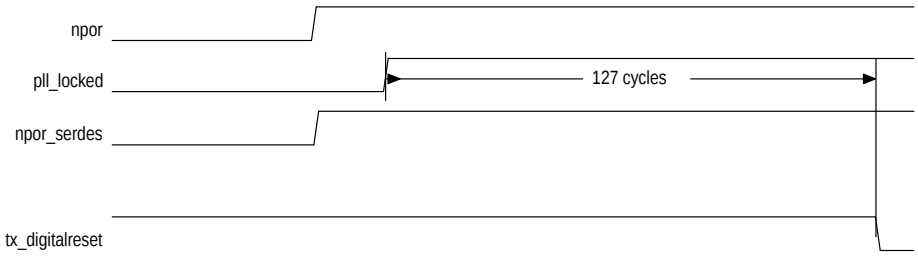


Reset Sequence for TX Transceiver

The following figure illustrates the TX transceiver reset sequence. As this figure illustrates, the RX transceiver reset includes the following steps:

1. After `npor` is deasserted, the core deasserts the `npor_serdes` input to the TX transceiver.
2. The SERDES reset controller waits for `pll_locked` to be stable for a minimum of 127 cycles before deasserting `tx_digitalreset`.

Figure 8-4: TX Transceiver Reset Sequence



For descriptions of the available reset *signals* refer to *Reset Signals, Status, and Link Training Signals*.

Related Information

Reset Signals, Status, and Link Training Signals on page 6-7

Clocks

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. The CDC synchronizer provides more flexibility for the user clock interface. Depending on parameters you specify, the core selects the appropriate `coreclkout_hip`. You can use these parameters

to enhance performance by running at a higher frequency for latency optimization or at a lower frequency to save power.

In accordance with the *PCI Express Base Specification* , you must provide a 100 MHz reference clock that is connected directly to the transceiver.

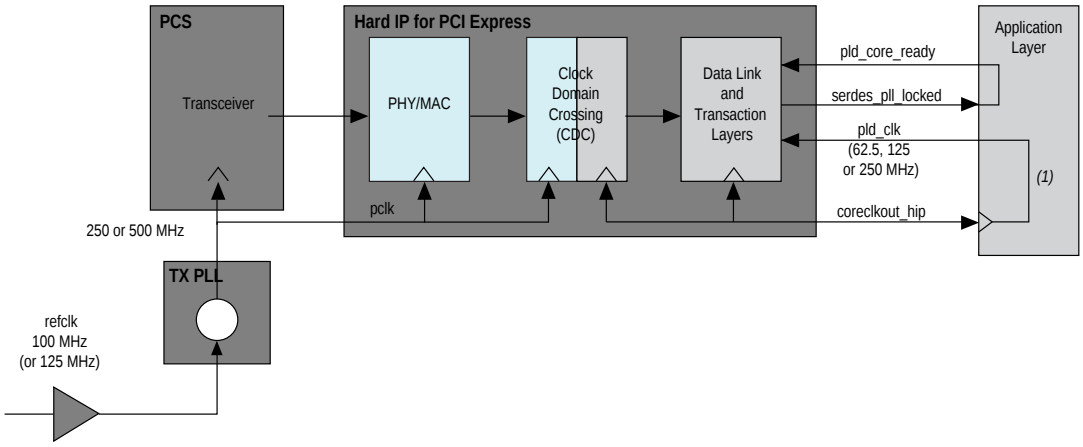
Related Information

PCI Express Base Specification 3.0

Arria 10 Hard IP for PCI Express Clock Domains

The following illustrates the clock domains when using `coreclkout_hip` to drive the Application Layer and the `pld_clk` of the Arria 10Hard IP for PCI Express IP Core. The Altera-provided example design connects `coreclkout_hip` to the `pld_clk`. However, this connection is not mandatory.

Figure 8-5: Clock Domains and Clock Generation for the Application Layer



As this figure indicates, the IP core includes the following three clock domains:

pclk

The transceiver derives `pclk` from the 100 MHz `refclk` signal that you must provide to the device.

The transitions between Gen1, Gen2, and Gen3 should be glitchless. `pclk` can be turned off for most of the 1 ms timeout assigned for the PHY to change the clock rate; however, `pclk` should be stable before the 1 ms timeout expires.

The following table shows the frequency of `pclk` for Gen1, Gen2, and Gen3 variants.

Table 8-1: pclk Clock Frequency

Data Rate	Frequency
Gen1	62.5 MHz
Gen2	125 MHz
Gen3	250 MHz

The CDC module implements the asynchronous clock domain crossing between the PHY/MAC `pclk` domain and the Data Link Layer `coreclk` domain. The transceiver `pclk` clock is connected directly to the Hard IP for PCI Express and does not connect to the FPGA fabric.

Related Information

PCI Express Base Specification 3.0

coreclkout_hip

The `coreclkout_hip` signal is derived from `pclk`. The following table lists frequencies for `coreclkout_hip`, which are a function of the link width, data rate, and the width of the Avalon-ST bus.

The frequencies and widths specified in this table 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 this table. (The Hard IP throttles the interface to achieve a lower throughput.)

Table 8-2: coreclkout_hip Values for All Parameterizations

Link Width	Max Link Rate	Avalon Interface Width(1)	coreclkout_hip
×1	Gen1	64	125 MHz
×1	Gen1	64	62.5 MHz (2)
×4	Gen1	64	125 MHz
×8	Gen1	64	250 MHz
×8	Gen1	128	125 MHz
×1	Gen2	64	125 MHz
×4	Gen2	64	250 MHz
×4	Gen2	128	125 MHz
×8	Gen2	128	250 MHz
×8	Gen2	256	125 MHz
×1	Gen3	64	125 MHz
×4	Gen3	128	250 MHz
×4	Gen3	256	125 MHz
×8	Gen3	256	250 MHz

pld_clk

`coreclkout_hip` can drive the Application Layer clock along with the `pld_clk` input to the Arria 10 Hard IP for PCI Express IP Core. The `pld_clk` can optionally be sourced by a different clock than

coreclkout_hip. The pld_clk minimum frequency cannot be lower than the coreclkout_hip frequency. Based on specific Application Layer constraints, a PLL can be used to derive the desired frequency.

Arria 10 Clock Summary

The following table summarizes the clocks for designs that include the Arria 10 Hard IP for PCI Express IP Core.

Table 8-3: Required Clocks

Name	Frequency	Clock Domain
Clock Used by the Arria 10 Hard IP for PCI Express IP Core		
coreclkout_hip	125 or 250 MHz	Avalon-ST interface between the Transaction and Application Layers.
pld_clk	Equal to or faster than coreclkout_hip	Application and Transaction Layers.
refclk	100 MHz	SERDES (transceiver). Dedicated free running input clock to the SERDES block.

Transaction Layer Protocol (TLP) Details

9

December 2013

UG-01145_avmm

Supported Message Types

INTX Messages

The following table describes the message INTX messages. For Endpoints, only INTA messages are generated.

Table 9-1: INTX Messages

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
INTX Mechanism						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	

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Deassert_INTC	Receive	Transmit	No	No	No	
Deassert_INTD	Receive	Transmit	No	No	No	

Power Management Messages

Table 9-2: Power Management Messages

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
PM_Active_State_Nak	TX	RX	No	Yes	No	—
PM_PME	RX	TX	No	No	Yes	—
PME_Turn_Off	TX	RX	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	RX	TX	No	No	Yes	—

Error Signaling Messages

Table 9-3: Error Signaling Messages

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
ERR_COR	RX	TX	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	RX	TX	No	Yes	No	—
ERR_FATAL	RX	TX	No	Yes	No	—

Locked Transaction Message

Table 9-4: Locked Transaction Message

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Unlock Message	Transmit	Receive	Yes	No	No	

Slot Power Limit Message

The *PCI Express Base Specification Revision*states that this message is not mandatory after link training.

Table 9-5: Slot Power Message

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Set Slot Power Limit	Transmit	Receive	No	Yes	No	In Root Port mode, through software.

Related Information

PCI Express Base Specification Revision 3.0

Vendor-Defined Messages

Table 9-6: Vendor-Defined Message

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Vendor Defined Type 0	Transmit Receive	Transmit Receive	Yes	No	No	
Vendor Defined Type 1	Transmit Receive	Transmit Receive	Yes	No	No	

Hot Plug Messages

Table 9-7: Locked Transaction Message

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Attention_indicator On	Transmit	Receive	No	Yes	No	As per the recommendations in the *PCI Express Base Specification Revision* , 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 (Endpoint only)	Receive	Transmit	No	No	Yes	—

Related Information

PCI Express Base Specification Revision 3.0

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 10 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 15–3 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.
- 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.
- The Type 1 Configuration TLPs are sent downstream on the PCI Express link. If the bus number of the Type 1 Configuration TLP matches the Secondary Bus Number register value in the Root Port Configuration Space, the TLP is converted to a Type 0 TLP.
- For more information on routing rules in Root Port mode, refer to *Section 7.3.3 Configuration Request Routing Rules* in the *PCI Express Base Specification* .

Related Information

PCI Express Base Specification Revision 3.0

Receive Buffer Reordering

The PCI, PCI-X and PCI Express protocols include ordering rules for concurrent TLPs. Ordering rules are necessary for the following reasons:

- To guarantee that TLP complete in the intended order
- To avoid deadlock

- To maintain computability with ordering used on legacy buses
- To maximize performance and throughput by minimizing read latencies and managing read/write ordering
- To avoid race conditions in systems that include legacy PCI buses by guaranteeing that reads to an address do not complete before an earlier write to the same address

PCI uses a strongly-ordered model with some exceptions to avoid potential deadlock conditions. PCI-X added a relaxed ordering (RO) bit in the TLP header. It is bit 5 of byte 2 in the TLP header, or the high-order bit of the `attributes` field in the TLP formats shown in Chapter A, Transaction Layer Packet (TLP) Header Formats. If this bit is set, relaxed ordering is permitted. If software can guarantee that no dependencies exist between pending transactions, it is safe to set the relaxed ordering bit.

The following table summarizes the ordering rules from the PCI specification. In this table, the entries have the following meanings:

- Columns represent the first transaction issued.
- Rows represent the next transaction.
- At each intersection, the implicit question is: should this row packet be allowed to pass the column packet? The following three answers are possible:
- Yes: the second transaction must be allowed to pass the first to avoid deadlock.
- Y/N: There are no requirements. A device may allow the second transaction to pass the first.
- No: The second transaction must not be allowed to pass the first.

The following table lists the transaction ordering rules. 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. 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. Endpoints, Switches, and Root Complex may allow Memory Write and Message Requests to pass Completions or be blocked by Completions. Memory Write and Message Requests can pass Completions traveling in the PCI Express to PCI directions to avoid deadlock. If the Relaxed Ordering attribute is not set, then a Read Completion cannot pass a previously enqueued Memory Write or Message Request. If the Relaxed Ordering attribute is set, then a Read Completion is permitted to pass a previously enqueued Memory Write or Message Request. Read Completion associated with different Read Requests are allowed to be blocked by or to pass each other. Read Completions for Request (same Transaction ID) must return in address order. Non-posted requests cannot pass other non-posted requests.

Table 9-8: Transaction Ordering Rules

Can the Row Pass the Column?		Posted Req		Non Posted Req				Completion	
		Memory Write or Message Req		Read Request		I/O or Cfg Write Req			
		Spec	Hard IP	Spec	Hard IP	Spec	Hard IP	Spec	Hard IP
P	Posted Req	No Y/N	No No	Yes	Yes	Yes	Yes	Y/N Yes	No No

Can the Row Pass the Column?		Posted Req		Non Posted Req				Completion	
		Memory Write or Message Req		Read Request		I/O or Cfg Write Req			
NP	Read Req	No	No	Y/N	No	Y/N	No	Y/N	No
	Non-Posted Req with data	No	No	Y/N	No	Y/N	No	Y/N	No
Cmpl	Completion	No Y/N	No No	Yes	Yes	Yes	Yes	Y/N No	No No
	I/O or Configu-ration Write Cmpl	Y/N	No	Yes	Yes	Yes	Yes	Y/N	No

The following are footnotes for the previous table:

1. Refers to the PCI Express Base Specification 3.0.
2. CfgRd0 can pass IORd or MRd.
3. CfgWr0 can IORd or MRd.
4. CfgRd0 can pass IORd or MRd
5. CfrWr0 can pass IOWr.

As the table above indicates, the RX datapath implements an 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 Req transactions made to the Application Layer interface so that only posted and completion transactions are presented to the Application Layer.

Note: 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.

Related Information

PCI Express Base Specification Revision 3.0

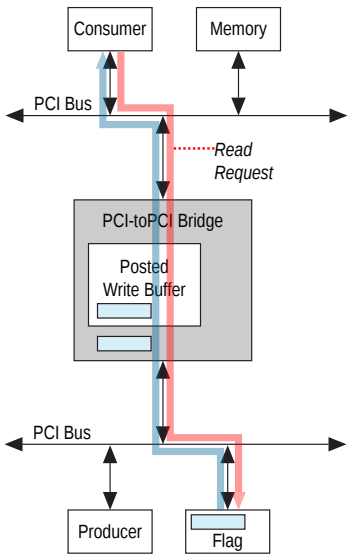
Using Relaxed Ordering

Transactions from unrelated threads are unlikely to have data dependencies. Consequently, you may be able to use relaxed ordering to improve system performance. The drawback is that only some transactions can

be optimized for performance. Complete the following steps to decide whether to enable relaxed ordering in your design:

1. Create a system diagram showing all PCI Express and legacy devices.
2. Analyze the relationships between the components in your design to identify the following hazards:
   a. Race conditions: A race condition exists if a read to a location can occur before a previous write to that location completes.The following figure shows a data producer and data consumer on opposite sides of a PCI-to-PCI bridge. The producer writes data to the memory through a PCI-to-PCI bridge. The consumer must read a flag to confirm the producer has written the new data into the memory before reading the data. However, because the PCI-to-PCI bridge includes a write buffer, the flag may indicate that it is safe to read data while the actual data remains in the PCI-to-PCI bridge posted write buffer.

Figure 9-1: Design Including Legacy PCI Buses Requiring Strong Ordering

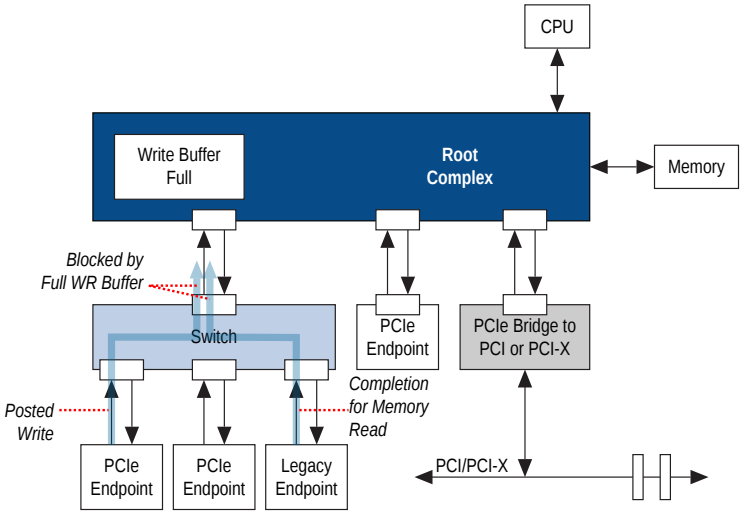


   b. A shared memory architecture where more than one thread accesses the same locations in memory.

   If either of these conditions exists, relaxed ordering will lead to incorrect results.

3. If your analysis determines that relaxed ordering does not lead to possible race conditions or read or write hazards, you can enable relaxed ordering by setting the RO bit in the TLP header. The following figure shows two PCIe Endpoints and Legacy Endpoint connected to a switch. The three PCIe Endpoints are not likely to have data dependencies. Consequently, it would be safe to set the relaxed ordering bit for devices connected to the switch. In this system, failing to enable relaxed ordering blocks a memory read to the Legacy Endpoint because an earlier posted write cannot complete because a write buffer is full.

Figure 9-2: PCI Express Design Using Relaxed Ordering



4. If your analysis indicates that you can enable relaxed ordering, simulate your system with and without relaxed ordering enabled. Compare the results and performance.
5. If relaxed ordering improves performance without introducing errors, you can enable it in your system.

Interrupts 10

December 2013

UG-01145_avmm

Interrupts for Endpoints Using the Avalon-MM Interface to the Application Layer

The PCI Express Avalon-MM bridge supports MSI or legacy interrupts. The completer only single dword variant includes an interrupt handler that implements both INTX and MSI interrupts. Support requires instantiation of the CRA slave module where the interrupt registers and control logic are implemented.

The PCI Express Avalon-MM bridge supports the Avalon-MM individual requests interrupt scheme: multiple input signals indicate incoming interrupt requests, and software must determine priorities for servicing simultaneous interrupts the Avalon-MM Arria 10 Hard IP for PCI Express receives.

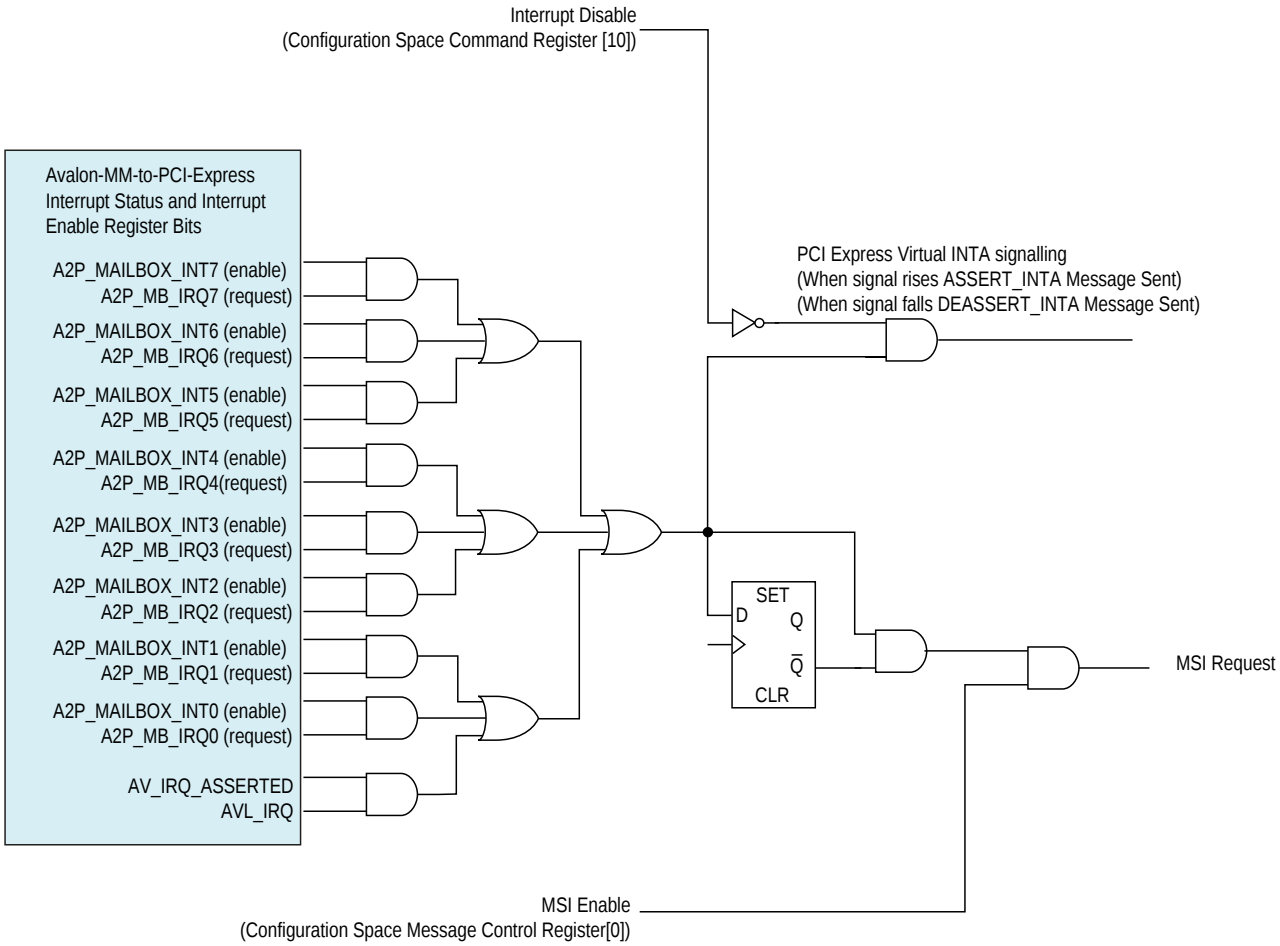
The RX master module port has as many as 16 Avalon-MM interrupt input signals (`RXmirq_irq[` *<n>* `:0]`, where *<n>* ≤15)). Each interrupt signal indicates a distinct interrupt source. Assertion of any of these signals, or a PCI Express mailbox register write access, sets a bit in the `Avalon-MM to PCI Express Interrupt Status` register. Multiple bits can be set at the same time; Application Layer software on the host side determines priorities for servicing simultaneous incoming interrupt requests. Each set bit in the `Avalon-MM to PCI Express Interrupt Status` interrupt status register generates a PCI Express interrupt, if enabled, when software determines its turn. Software can enable the individual interrupts by writing to the *Avalon-MM to PCI Express Interrupt Enable Register* through the CRA slave.

When any interrupt input signal is asserted, the corresponding bit is written in the *Avalon-MM to PCI Express Interrupt Status Register*. Software reads this register and decides priority on servicing requested interrupts.

After servicing the interrupt, software must clear the appropriate serviced interrupt `status` bit and ensure that no other interrupts are pending. For interrupts caused by *Avalon-MM to PCI Express Interrupt Status Register (0x0040)* mailbox writes, the status bits should be cleared in the *Avalon-MM to PCI Express Interrupt Status Register 0x0040*. For interrupts due to the incoming interrupt signals on the Avalon-MM interface, the interrupt status should be cleared in the Avalon-MM component that sourced the interrupt. This sequence prevents interrupt requests from being lost during interrupt servicing.

The following figure shows the logic for the entire interrupt generation process.

Figure 10-1: Avalon-MM Interrupt Propagation to the PCI Express Link



Related Information

- **Avalon-MM to PCI Express Interrupt Enable Registers** on page 7-17
- **Avalon-MM to PCI Express Interrupt Status Registers** on page 7-16

Enabling MSI or Legacy Interrupts

The PCI Express Avalon-MM bridge selects either MSI or legacy interrupts automatically based on the standard interrupt controls in the PCI Express Configuration Space registers. Software can write the `Interrupt Disable` bit, which is bit 10 of the `Command` register (at Configuration Space offset 0x4) to disable legacy interrupts. Software can write the `MSI Enable` bit, which is bit 0 of the `MSI Control Status` register in the MSI capability register (bit 16 at configuration space offset 0x50), to enable MSI interrupts.

Software can only enable one type of interrupt at a time. However, to change the selection of MSI or legacy interrupts during operation, software must ensure that no interrupt request is dropped. Therefore, software must first enable the new selection and then disable the old selection. To set up legacy interrupts, software

must first clear the Interrupt Disable bit and then clear the MSI enable bit. To set up MSI interrupts, software must first set the MSI enable bit and then set the Interrupt Disable bit.

Generation of Avalon-MM Interrupts

Generation of Avalon-MM interrupts requires the instantiation of the CRA slave module where the interrupt registers and control logic are implemented. The CRA slave port has an Avalon-MM Interrupt output signal, cra_Irq_irq. A write access to an Avalon-MM mailbox register sets one of the P2A_MAILBOX_INT <*n*> bits in the *Avalon-MM to PCI Express Interrupt Status Register 0x0040* and asserts the cra_Irq_o or cra_Irq_irq output, if enabled. Software can enable the interrupt by writing to the *INT-X Interrupt Enable Register for Endpoints 0x3070*through the CRA slave. After servicing the interrupt, software must clear the appropriate serviced interrupt status bit in the PCI-Express-to-Avalon-MM Interrupt Status register and ensure that no other interrupt is pending.

Related Information

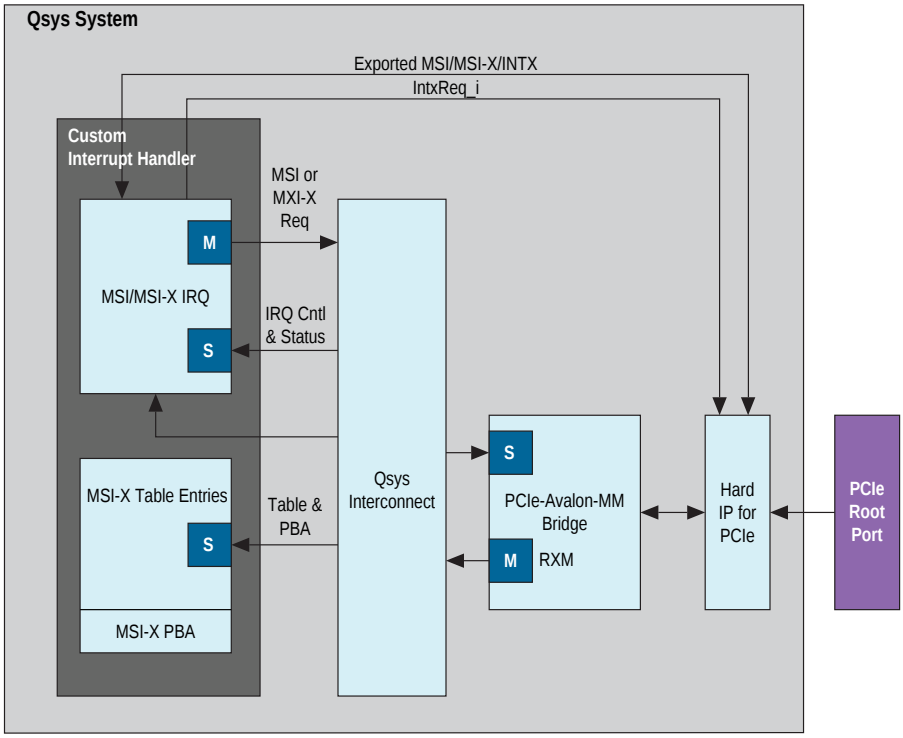
- **Avalon-MM to PCI Express Interrupt Status Registers** on page 7-16
- **PCI Express to Avalon-MM Interrupt Status and Enable Registers for Endpoints** on page 7-20

Interrupts for End Points Using the Avalon-MM Interface with Multiple MSI/MSI-X Support

If you select **Enable multiple MSI/MSI X support** under the **Avalon MM System Settings** banner in the GUI, the Hard IP for PCI Express exports the MSI, MSI-X, and INTx interfaces to the Application Layer. The Application Layer must include a Custom Interrupt Handler to send interrupts to the Root Port. You must design this Custom Interrupt Handler. The following figure provides a an overview of the logic for the Custom Interrupt Handler. The Custom Interrupt Handler should include hardware to perform the following tasks:

- An MSI/MXI-X IRQ Avalon-MM Master port to drive MSI or MSI-X interrupts as memory writes to the PCIe Avalon-MM Bridge.
- A legacy interrupt signal, IntxReq_i, to drive legacy interrupts from the MSI/MSI-X IRQ module to the Hard IP for PCI Express.
- An MSI/MSI-X Avalon-MM Slave port to receive interrupt control and status from PCIe Root Port.
- An MSI-X table to store the MSI-X table entries. The PCIe Root Port sets up this table.

Figure 10-2: Block Diagram for Custom Interrupt Handler



Refer to *Interrupts for Endpoints* for the definitions of MSI, MSI-X and INTx buses.

For more information about implementing MSI or MSI-X interrupts, refer to the *PCI Local Bus Specification, Revision 2.3, MSI-X ECN*.

Related Information

PCI Local Bus Specification, Revision 2.3, MSI

Throughput Optimization 11

December 2013

UG-01145_avmm Subscribe Send Feedback

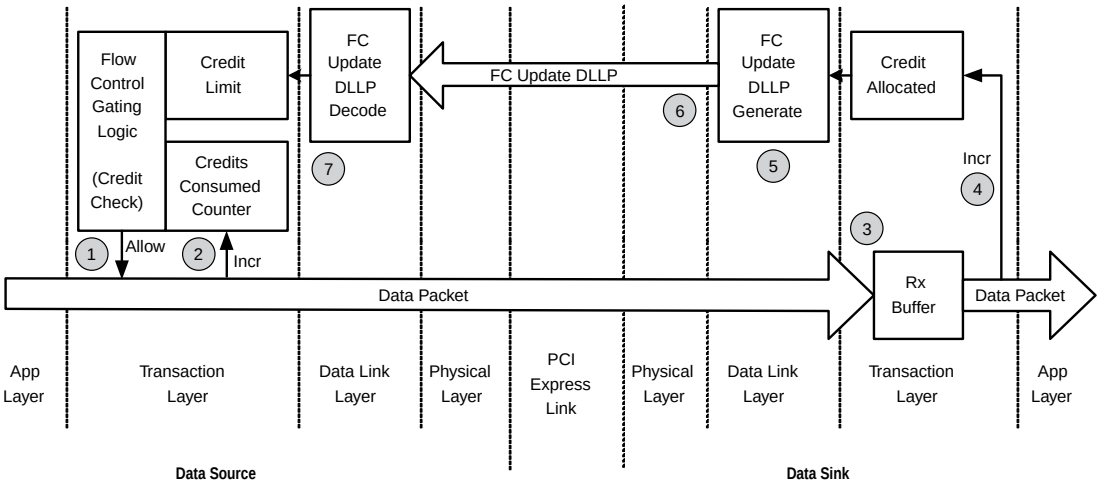
The *PCI Express Base Specification* defines a flow control mechanism to ensure efficient transfer of TLPs.

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 received 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 in the figure 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` register 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 `MAX_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 `MAX_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.

Note: You must keep track of the credits consumed by the Application Layer.

Throughput of Posted Writes

The throughput of posted writes is limited primarily by the Flow Control Update loop shown in the previous figure. 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.

The figure below shows the main components of the Flow Control Update loop with two communicating PCI Express ports:

- Write Requester
- Write Completer

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.

Related Information

PCI Express Base Specification 3.0

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 10 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 for Completions 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*. 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 10 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.

Error Handling 12

December 2013

UG-01145_avmm

Each PCI Express compliant device must implement a basic level of error management and can optionally implement advanced error management. The Arria 10Hard 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* defines three types of errors, outlined in the following table.

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.

Related Information

PCI Express Base Specification 3.0

Physical Layer Errors

The following table describes errors detected by the Physical Layer. Physical Layer error reporting is optional in the *PCI Express Base Specification Revision* .

Table 12-2: Errors Detected by the Physical Layer

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 Error101: Elastic Buffer Overflow110: Elastic Buffer Underflow111: Disparity Error • Deskew error caused by overflow of the multilane deskew FIFO. • Control symbol received in wrong lane.

Data Link Layer Errors

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.

Related Information

Data Link Layer Errors on page 12-2

Transaction Layer Errors

Table 12-4: Errors Detected by the Transaction Layer

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* 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 and function number. • 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 BAR match. • 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`) In all cases the TLP is deleted in the Hard IP block and not presented to the Application Layer. If the TLP is a

Error	Type	Description
		non-posted request, the Hard IP block generates a completion with Unsupported Request status.
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
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.

Error	Type	Description
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)	This error occurs when a component does not receive update flow control credits with the 200 µs limit.

Error	Type	Description
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. • 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:

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

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 3.0* 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. The following table 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.

Related Information

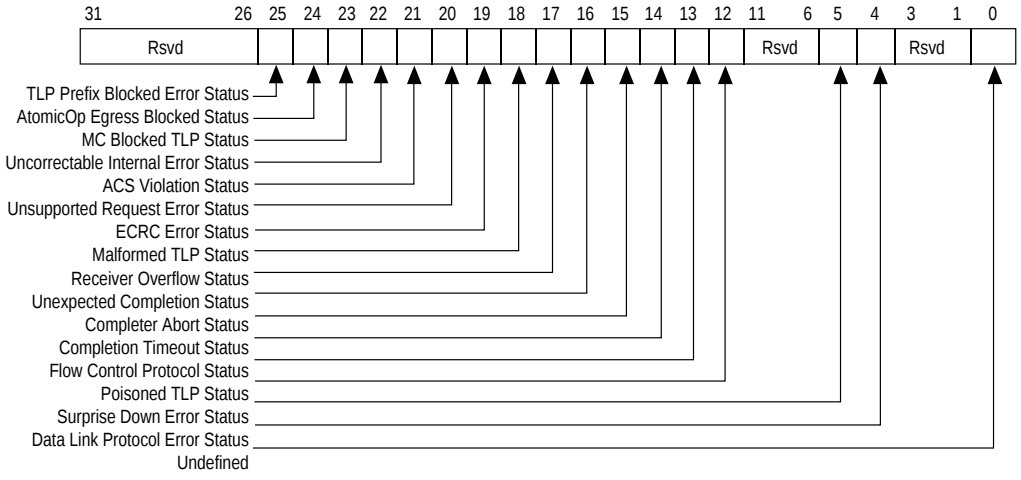
PCI Express Base Specification 3.0

Uncorrectable and Correctable Error Status Bits

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

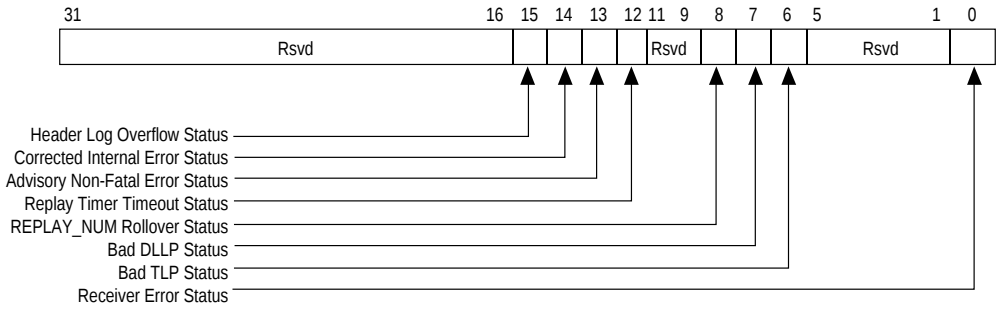
The following figure illustrates the Uncorrectable Error Status register. The default value of all the bits of this register is 0. An error status bit that is set indicates that the error condition it represents has been detected. Software may clear the error status by writing a 1 to the appropriate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

Figure 12-1: Uncorrectable Error Status Register



The following figure 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



Design Implementation 13

December 2013

UG-01145_avmm

Completing your design includes additional steps to specify analog properties, pin assignments, and timing constraints.

Making Analog QSF Assignments Using the Assignment Editor

You specify the analog parameters using the Quartus II Assignment Editor, the Pin Planner, or through the Quartus II Settings File **.(qsf)**.

The Quartus II software provides default values for analog parameters. You can change the defaults using Assignment Editor or the Pin Planner. You can also edit your **.qsf** directly or by typing commands in the Quartus II Tcl Console.

The following example shows how to change the value of the voltages required:

1. On the Assignments menu, select **Assignment Editor**. The Assignment Editor appears.
2. Complete the following steps for each pin requiring the V_{CCR_GXB} and V_{CCT_GXB} voltage:
   a. Double-click in the **Assignment Name** column and scroll to the bottom of the available assignments.
   b. Select **VCCR_GXB/VCCT_GXB Voltage**.
   c. In the **Value** column, select **.0V** from the list.

The Quartus II software adds these instance assignments commands to the **.qsf** file for your project.

You can also enter these commands at the Quartus II Tcl Console. For example, the following command sets the `XCVR_VCCR_VCCT_VOLTAGE` to 1.0 V for the pin specified:

```
set_instance_assignment -name XCVR_VCCR_VCCT_VOLTAGE 1_0V to “pin”
```

Making Pin Assignments

Before running Quartus II compilation, use the **Pin Planner** to assign I/O standards to the pins of the device. Complete the following steps to bring up the **Pin Planner** and assign the 1.5-V pseudo-current mode logic (PCML) I/O standard to the serial data input and output pins:

1. On the Quartus II **Assignments** menu, select **Pin Planner**. The **Pin Planner** appears.
2. In the **Node Name** column, locate the PCIe serial data pins.
3. In the **I/O Standard** column, double-click the right-hand corner of the box to bring up a list of available I/O standards.
4. Select **1.5 V PCML I/O** standard.

Note: The Arria 10 Hard IP for PCI Express IP Core automatically assigns other required PMA analog settings, including 100 ohm internal termination.

SDC Timing Constraints

Example 13-1: SDC Timing Constraints Required for the Arria 10 Hard IP for PCIe and Design Example

```
# Constraints required for the Arria 10 Hard IP for PCI Express
# derive_pll_clock is used to calculate all clock derived
# from PCIe refclk. It must be applied once across all
# of the SDC files used in a project
derive_pll_clocks -create_base_clocks
derive_clock_uncertainty

##############################################################################

# Hard IP testin pins SDC constraints
set_false_path -from [get_pins -compatibility_mode *hip_ctrl*]
```

In the example above, you should only apply the first two constraints, to derive PLL clocks and clock uncertainty, once across all of the SDC files in your project. Differences between Fitter timing analysis and TimeQuest timing analysis arise if these constraints are applied more than once.

When implementing the Hard IP for PCI Express in Arria 10 devices, the following two additional **.sdc** files are required:

- **altera_xcvr_native_a10_b2.sdc**
- **altera_xcvr_native_a10_false_paths.sdc**

These files are automatically generated when you generate the Arria 10 Hard IP for PCI Express IP Core.

Optional Features 14

December 2013

UG-01145_avmm

Subscribe Send Feedback

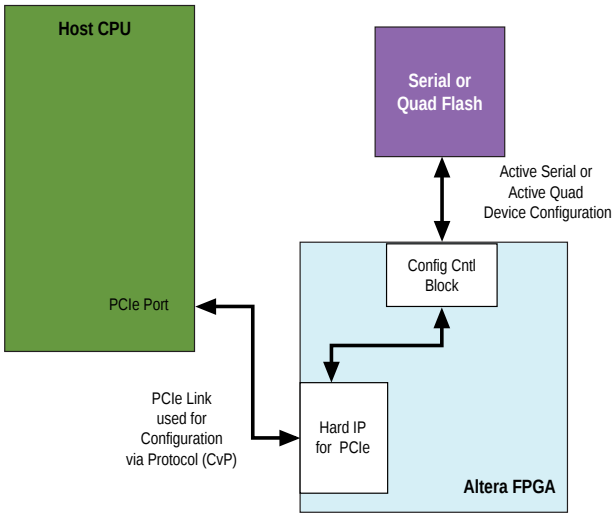
Configuration via Protocol (CvP)

The Hard IP for PCI Express architecture introduces has an option for sequencing the processes that configure the FPGA and initializes the PCI Express link. In prior devices, a single Program Object File (**.pof**) programmed the I/O ring and FPGA fabric before the PCIe link training and enumeration began. In Arria 10, the **.pof** file is divided into two parts:

- The I/O bitstream contains the data to program the I/O ring, the Hard IP for PCI Express, and other elements that are considered part of the periphery image.
- The core bitstream contains the data to program the FPGA fabric.

In Arria 10 devices, when you select the CvP design flow, the I/O ring and PCI Express link are programmed first, allowing the PCI Express link to reach the L0 state and begin operation independently, before the rest of the core is programmed. After the PCI Express link is established, it can be used to program the rest of the device. The following figure shows the blocks that implement CvP.

Figure 14-1: CvP in Arria 10 Devices



CvP has the following advantages:

- Provides a simpler software model for configuration. A smart host can use the PCIe protocol and the application topology to initialize and update the FPGA fabric.
- Improves security for the proprietary core bitstream.
- Reduces system costs by reducing the size of the flash device to store the **.pof**.
- Facilitates hardware acceleration.
- May reduce system size because a single CvP link can be used to configure multiple FPGAs.
- CvP is available for Gen1, Gen2, and Gen3 configurations.

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 forward the TLP with ECRC to the RX port of the Application Layer. When using ECRC forwarding mode, the ECRC check and generation 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** heading using the GUI 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*.

Related Information

PCI Express Base Specification 3.0

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.

The following table summarizes the RX ECRC functionality for all possible conditions.

Table 14-1: ECRC Operation on RX Path

ECRC Forwarding	ECRC Check Enable [2]	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
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

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. The following table summarizes the TX ECRC generation and forwarding. All unspecified cases are unsupported and the behavior of the Hard IP is unknown.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. Create related link

[2] The ECRC Check Enable field is in the Configuration Space Advanced Error Capabilities and Control Register.

Table 14-2: ECRC Generation and Forwarding on TX Path

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

ECRC Forwarding	ECRC Generation Enable [3]	TLP on Application	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	

[3] The ECRC Generation Enable field is in the Configuration Space Advanced Error Capabilities and Control Register.

Hard IP Reconfiguration 15

December 2013

UG-01145_avmm

The Arria 10 Hard IP for PCI Express reconfiguration block allows you to dynamically change the value of configuration registers that are *read-only*. You access this block using its Avalon-MM slave interface. You must enable this optional functionality by turning on **Enable Hard IP Reconfiguration** in the GUI. For a complete description of the signals in this interface, refer to *Hard IP Reconfiguration Interface*.

The Hard IP reconfiguration block provides access to *read-only* configuration registers, including Configuration Space, Link Configuration, MSI and MSI-X capabilities, Power Management, and Advanced Error Reporting (AER). This interface does not support simulation.

The procedure to dynamically reprogram these registers includes the following three steps:

1. Bring down the PCI Express link by asserting the `hip_reconfig_rst_n` reset signal, if the link is already up. (Reconfiguration can occur before the link has been established.)
2. Reprogram configuration registers using the Avalon-MM slave Hard IP reconfiguration interface.
3. Release the `npor` reset signal.

Note: You can use the LMI interface to change the values of configuration registers that are *read/write* at run time. For more information about the LMI interface, refer to *LMI Signals*.

Reconfigurable Read-Only Registers in the Hard IP for PCI Express

The following table lists all of the registers that you can update using the PCI Express reconfiguration block interface.

Address	Bits	Description	Default Value
0x00	0	When 0, PCIe reconfig mode is enabled. When 1, PCIe reconfig mode is disabled and the original read only register values set in the programming file used to configure the device are restored.	b'1
0x01-0x88	—	Reserved.	—
0x89	15:0	Vendor ID.	0x1172
0x8A	15:0	Device ID.	0x0001
0x8B	7:0	Revision ID.	0x01

Address	Bits	Description	Default Value
0x8C	15:0	Class code[23:8].	—
0x8D	15:0	Subsystem vendor ID.	0x1172
0x8E	15:0	Subsystem device ID.	0x0001
0x8F	—	Reserved.	—
0x90	0	Advanced Error Reporting.	b'0
	3:1	Low Priority VC (LPVC).	b'000
	7:4	VC arbitration capabilities.	b'00001
	15:8	Reject Snoop Transaction.	b'00000000
0x91	2:0	Max payload size supported. The following are the defined encodings: • 000: 128 bytes max payload size • 001: 256 bytes max payload size • 010: 512 bytes max payload size • 011: 1024 bytes max payload size • 100: 2048 bytes max payload size • 101: 4096 bytes max payload size • 110: Reserved, • 111: Reserved	b'010
	3	Surprise Down error reporting capabilities. (Available in *PCI Express Base Specification Revision 1.1* compliant Cores, only.) Downstream Port. This bit must be set to 1 if the component supports the optional capability of detecting and reporting a Surprise Down error condition. Upstream Port. For upstream ports and components that do not support this optional capability, this bit must be hardwired to 0. (Available in *PCI Express Base Specification Revision 1.1* compliant Cores, only.) Downstream Port: This bit must be set to 1 if the component supports the optional capability of reporting the DL_ Active state of the Data Link Control and Management state machine. Upstream Port: For upstream ports and components that do not support this optional capability, this bit must be hardwired to 0. Extended TAG field supported.	b'0
	4	(Available in *P CI Express Base Specification Revision 1.1* compliant Cores, only.) Downstream Port: This bit must be set to 1 if the component supports the optional capability of reporting the DL_Active state of the Data Link Control and Management state machine. Upstream Port: For upstream ports and components that do not support this optional capability, this bit must be hardwired to 0.	b'0

Address	Bits	Description	Default Value
	5	Extended TAG field supported	b'0
	8:6	Endpoint L0s acceptable latency. The following encodings are defined: • b'000 - Maximum of 64 ns • b'001 - Maximum of 128 ns • b'010 - Maximum of 256 ns • b'011 - Maximum of 512 ns • b'100 - Maximum of 1 µs • b'101 - Maximum of 2 µs • b'110 - Maximum of 4 µs • b'111- No limit.	b'000
	11:9	Endpoint L1 acceptable latency. The following encodings are defined: • b'000 - Maximum of 1 µs • b'001 - Maximum of 2 µs • b'010 - Maximum of 4 µs • b'011 - Maximum of 8 µs • b'100 - Maximum of 16 µs • b'101 - Maximum of 32 µs • b'110 - Maximum of 64 µs • b'111 - No limit.	b'000
	14:12	These bits record the presence or absence of the attention and power indicators. • [0]: Attention button present on the device. • [1]: Attention indicator present for an endpoint. • [2]: Power indicator present for an endpoint.	b'1
	15	Role-Based error reporting. (Available in *PCI Express Base Specification Revision 1.1* compliant Cores only.)	—

Address	Bits	Description	Default Value
0x92	1:0	Slot Power Limit Scale.	b'00
	7:2	Max Link width.	b'000100
	9:8	L0s Active State power management support. L1 Active State power management support.	b'01
	15:10	L1 exit latency common clock. L1 exit latency separated clock. The following encodings are defined: • b'000 - Less than 1 µs. • b'001 - 1 µs to less than 2 µs • b'010 - 2 µs to less than 4 µs • b'011 - 4 µs to less than 8 µs • b'100 - 8 µs to less than 16 µs • b'101 - 16 µs to less than 32 µs • b'110 - 32 µs to 64 µs • b'111 - More than 64 µs	b'000000
0x93	0	Attention button implemented on the chassis.	b'0000000
	1	Power controller present.	
	2	Manually Operated Retention Latch (MRL) sensor present.	
	3	Attention indicator present for a root port, switch, or bridge.	
	4	Power indicator present for a root port, switch, or bridge.	
	5	Hot-plug surprise: When this bit set to1, a device can be removed from this slot without prior notification.	
	6	Hot-plug capable.	
	9:7	Reserved.	b'000
	15:10	Slot Power Limit Value.	b'00000000
0x94	1:0	Reserved.	-—
	2	Electromechanical Interlock present (Available in *PCI Express Base Specification Revision 1.1* compliant IP cores only.)	b'0
	15:3	Physical Slot Number (if slot implemented). This signal indicates the physical slot number associated with this port. It must be unique within the fabric.	b'0
0x95	7:0	NFTS_SEPCLK. The number of fast training sequences for the separate clock.	b'10000000
	15:8	NFTS_COMCLK. The number of fast training sequences for the common clock.	b'10000000
0x96	3:0	Completion timeout ranges. The following encodings are defined: • b'0001: range A	b'0000

Address	Bits	Description	Default Value
		• b'0010: range B • b'0011: range A&B • b'0110: range B&C • b'0111: range A,B&C • b'1110: range B,C&D • b'1111: range A,B,C&D All other values are reserved.	
	4	Completion Timeout supported. 0: completion timeout disable not supported 1: completion timeout disable supported	b'0
	7:5	Reserved.	b'0
	8	ECRC generate.	b'0
	9	ECRC check.	b'0
	10	No command completed support. (available only in *PCI Express Base Specification Revision 1.1* compliant Cores)	b'0
	13:11	Number of functions MSI capable. • b'000: 1 MSI capable • b'001: 2 MSI capable • b'010: 4 MSI capable • b'011: 8 MSI capable • b'100: 16 MSI capable • b'101: 32 MSI capable	b'010
	14	MSI 32/64-bit addressing mode. b'0: 32 bits only. b'1: 32 or 64 bits	b'1
	15	MSI per-bit vector masking (read-only field).	b'0
0x97	0	Function supports MSI.	b'1
	3:1	Interrupt pin.	b'001
	5:4	Reserved.	b'00
	6	Function supports MSI-X.	b'0
	15:7	MSI-X table size	b'0
0x98	1:0	Reserved.	—
	4:2	MSI-X Table BIR.	b'0
	15:5	MIS-X Table Offset.	b'0
0x99	15:10	MSI-X PBA Offset.	b'0
0x9A	15:0	Reserved.	b'0
0x9B	15:0	Reserved.	b'0
0x9C	15:0	Reserved.	b'0

Address	Bits	Description	Default Value
0x9D	15:0	Reserved.	b'0
0x9E	3:0	Reserved.	—
	7:4	Number of EIE symbols before NFTS.	b'0100
	15:8	Number of NFTS for separate clock in Gen2 rate.	b'11111111
0x9F	7:0	Number of NFTS for common clock in Gen2 rate.	b'11111111
	8	Selectable de-emphasis.	b'0
	12:9	PCIe Capability Version. • b'0000: Core is compliant to PCIe Specification 1.0a or 1.1 • b'0001: Core is compliant to PCIe Specification 1.0a or 1.1 • b'0010: Core is compliant to PCIe Specification 2.0 • b'0010: Core is compliant to PCIe Specification 3.0	b'0010
	15:13	L0s exit latency for common clock. • Gen1: (*N_FTS* (of separate clock) + 1 (for the SKIPOS)) * 4 * 10 * *UI* (*UI* = 0.4 ns). • Gen2: [(*N_FTS2* (of separate clock) + 1 (for the SKIPOS)) * 4 + 8 (max number of received *EIE*)] * 10 * *UI* (*UI* = 0.2 ns).	b'110
0xA0	2:0	L0s exit latency for separate clock. 1. Gen1: (*N_FTS* (of separate clock) + 1 (for the SKIPOS)) * 4 * 10 * *UI* (*UI* = 0.4 ns). 2. Gen2: [(*N_FTS2* (of separate clock) + 1 (for the SKIPOS)) * 4 + 8 (max number of received *EIE*)] * 10 * *UI* (*UI* = 0.2 ns). • b'000 - Less than 64 ns. • b'001 - 64 ns to less than 128 ns • b'010 - 128 ns to less than 256 ns • b'011 - 256 ns to less than 512 ns • b'100 - 512 ns to less than 1 µs • b'101 - 1 µs to less than 2 µs • b'110 - 2 µs to 4 µs • b'111 - More than 4 µs	b'110
	15:3	Reserved.	0x0000

Address	Bits	Description	Default Value
0xA1		**BAR0[31:0]**	
	0	BAR0[0]: I/O Space.	b'0
	2:1	BAR0[2:1]: Memory Space. The following encodings are defined: • 2'b10: 64-bit address • 2b'00: 32-bit address	b'10
	3	BAR0[3]: Prefetchable.	b'1
		BAR0[31:4]: Bar size mask.	0xFFFFFFF
	15:4	BAR0[15:4].	b'0
0xA2	15:0	BAR0[31:16].	b'0
0xA3		**BAR1[63:32]**	b'0
	0	BAR1[32]: I/O Space.	b'0
	2:1	BAR1[34:33]: Memory Space (see bit settings for BAR0).	b'0
	3	BAR1[35]: Prefetchable.	b'0
		BAR1[63:36]: Bar size mask.	b'0
	15:4	BAR1[47:36].	b'0
0xA4	15:0	BAR1[63:48].	b'0
0xA5		BAR2[95:64]:	b'0
	0	BAR2[64]: I/O Space.	b'0
	2:1	BAR2[66:65]: Memory Space (see bit settings for BAR0).	b'0
	3	BAR2[67]: Prefetchable.	b'0
		BAR2[95:68]: Bar size mask.	b'0
	15:4	BAR2[79:68].	b'0
0xA6	15:0	BAR2[95:80].	b'0
0xA7		BAR3[127:96].	b'0
	0	BAR3[96]: I/O Space.	b'0
	2:1	BAR3[98:97]: Memory Space (see bit settings for BAR0).	b'0
	3	BAR3[99]: Prefetchable.	b'0
		BAR3[127:100]: Bar size mask.	b'0
	15:4	BAR3[111:100].	b'0
0xA8	15:0	BAR3[127:112].	b'0

Address	Bits	Description	Default Value
0xA9		BAR4[159:128].	b'0
	0	BAR4[128]: I/O Space.	b'0
	2:1	BAR4[130:129]: Memory Space (see bit settings for BAR0).	b'0
	3	BAR4[131]: Prefetchable.	b'0
		BAR4[159:132]: Bar size mask.	b'0
	15:4	BAR4[143:132].	b'0
0xAA	15:0	BAR4[159:144].	b'0
0xAB		BAR5[191:160].	b'0
	0	BAR5[160]: I/O Space.	b'0
	2:1	BAR5[162:161]: Memory Space (see bit settings for BAR0).	b'0
	3	BAR5[163]: Prefetchable.	b'0
		BAR5[191:164]: Bar size mask.	b'0
	15:4	BAR5[175:164].	b'0
0xAC	15:0	BAR5[191:176].	b'0
0xAD	15:0	Expansion BAR[223:192]: Bar size mask. Expansion BAR[207:192].	b'0
0xAE	15:0	Expansion BAR[223:208].	b'0
0xAF	1:0	IO. • 00: no IO windows. • 01: IO 16 bit. • 11: prefetchable 64. • 11: IO 32-bit.	b'0
	3:2	Prefetchable. • 00: not implemented. • 01: prefetchable 32 • 11: prefetchable 64 • b'101: 32 MSI capable.	b'0
	15:4	Reserved.	—

Address	Bits	Description	Default Value
0xB0	5:0	Reserved	—
	6	Selectable de-emphasis, operates as specified in the *PCI Express Base Specification* when operating at the 5.0GT/s rate: • 1: 3.5 dB • 0: -6 dB This setting has no effect when operating at the 2.5GT/s rate.	
	9:7	Transmit Margin. Directly drives the transceiver `tx_pipemargin` bits.	
0xB1-FF	—	Reserved.	

Related Information

- **Hard IP Reconfiguration Interface** on page 6-9
- **PCI Express Base Specification 3.0**

Debugging 16

December 2013

UG-01145_avmm

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.

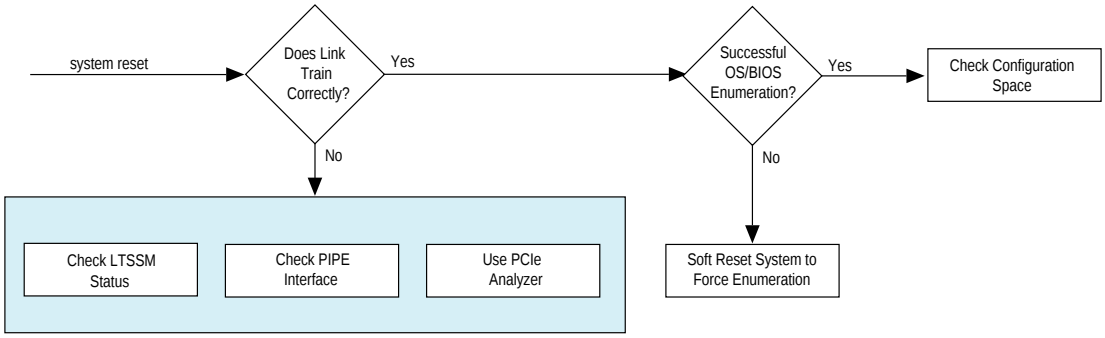
Hardware Bring-Up Issues

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

1. System reset
2. Link 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 the following figure.

Figure 16-1: Debugging Link Training Issues



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

You can use SignalTap II Embedded Logic Analyzer to diagnose the LTSSM state transitions that are occurring and the PIPE interface. 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 Signals, Status, and Link Training Signals*. 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 the cause.

Related Information

Reset Signals, Status, and Link Training Signals on page 6-7

Debugging Link that Fails To Reach L0

The following table describes possible causes of the failure to reach L0.

Table 16-1: Link Training Fails to Reach L0

Possible Causes	Symptoms and Root Causes	Workarounds and Solutions
Link fails the Receiver Detect sequence.	LTSSM toggles between Detect.Quiet(0) and Detect.Active(1) states	Check the following termination settings: • For Gen1 and Gen2, the *PCI Express Base Specification, Rev 3.0.* states a range of 0.075 µF–0.265 µF for on-chip termination (OCT). • For Gen3, the *PCI Express Base Specification, Rev 3.0.* states a range of 0.176 µF–0.265 µF for OCT. • Altera uses 0.22 µF terminations to ensure compliance across all data rates. • Link partner RX pins must also have appropriate values for terminations.

Possible Causes	Symptoms and Root Causes	Workarounds and Solutions
Link fails with LTSSM stuck in Detect.Active state (1)	This behavior may be caused by a PMA issue if the host interrupts the Electrical Idle state as indicated by high to low transitions on the RxElecIdle (`rxelecidle`) signal when TxDetectRx=0 (`txdetectrx0`) at PIPE interface. Check if OCT is turned off by a Quartus Settings File (**.qsf**) command. PCIe requires that OCT must be used for proper Receiver Detect with a value of 100 Ohm. You can debug this issue using SignalTap II and oscilloscope.	For Arria 10 devices, a workaround is implemented in the reset sequence.
Link fails with the LTSSM toggling between: Detect.Quiet (0), Detect.Active (1), and Polling.Active (2) , or: Detect.Quiet (0), Detect.Active (1), and Polling.Configuration (4)	On the PIPE interface extracted from the `test_out` bus, confirm that the Hard IP for PCI Express IP Core is transmitting valid TS1 in the Polling.Active(2) state or TS1 and TS2 in the Polling.Configuration (4) state on txdata0. The Root Port should be sending either the TS1 Ordered Set or a compliance pattern as seen on `rxdata0`. These symptoms indicate that the Root Port did not receive the valid training Ordered Set from Endpoint because the Endpoint transmitted corrupted data on the link. You can debug this issue using SignalTap II. Refer to *PIPE Interface Signals* for a list of the `test_out` bus signals.	The following are some of the reasons the Endpoint might send corrupted data: • Signal integrity issues. Measure the TX eye and check it against the eye opening requirements in the PCI Express Base Specification, Rev 3.0. Adjust the transceiver pre-emphasis and equalization settings to open the eye. • Bypass the Transceiver Reconfiguration Controller IP Core to see if the link comes up at the expected data rate without this component. If it does, make sure the connection to Transceiver Reconfig Controller IP Core is correct.

Possible Causes	Symptoms and Root Causes	Workarounds and Solutions
Link fails due to unstable `rx_signaldetect`	Confirm that `rx_signaldetect` bus of the active lanes is all 1's. If all active lanes are driving all 1's, the LTSSM state machine toggles between Detect.Quiet(0), Detect.Active(1), and Polling.Active(2) states.	This issue may be caused by mismatches between the expected power supply to RX side of the receiver and the actual voltage supplied to the FPGA from your boards. If your PCB drives VCCT/VCCR with 1.0 V, you must apply the following command to both P and N pins of each active channel to override the default setting of 0.85 V. `set_instance_assignment -name XCVR_VCCR_VCCT_VOLTAGE 1_0V –to "pin"` Substitute the pin names from your design for `"pin"`.
Link fails because the LTSSM state machine enters Compliance	Confirm that the LTSSM state machine is in Polling.Compliance(3) using SignalTap II.	Possible causes include the following: • Setting `test_in[6]`=1 forces entry to Compliance mode when a timeout is reached in the Polling.Active state. • Differential pairs are incorrectly connected to the pins of the device. For example, the Endpoint's TX signals are connected to the RX pins and the Endpoint's RX signals are to the TX pins.
Link fails because LTSSM state machine unexpectedly transitions to Recovery	A framing error is detected on the link causing LTSSM to enter the Recovery state.	In simulation, set `test_in[1]`=1 to speed up simulation. This solution only solves this problem for simulation. For hardware, customer must set `test_in[1]`=0.

Recommended Reset Sequence to Avoid Link Training Issues

Successful link training can only occur after the FPGA is configured and the Transceiver Reconfiguration Controller IP Core has dynamically reconfigured SERDES analog settings to optimize signal quality. For designs using CvP, link training occurs after configuration of the I/O ring and Hard IP for PCI Express IP Core. Refer to *Reset Sequence for Hard IP for PCI Express IP Core and Application Layer*for a description of

the key signals that reset, control dynamic reconfiguration, and link training. Successful reset sequence includes the following steps:

1. Wait until the FPGA is configured as indicated by the assertion of `CONF IG_DONE` from the FPGA block controller.
2. Deassert the `mgmt_rst_reset` input to the Transceiver Reconfiguration Controller IP Core.
3. Wait for `tx_cal_busy` and `rx_cal_busy` SERDES outputs to be deasserted.
4. Deassert `pin_perst` to take the Hard IP for PCIe out of reset. For plug-in cards, the minimum assertion time for `pin_perst` is 100 ms. Embedded systems do not have a minimum assertion time for `pin_perst`.
5. Wait for the `reset_status` output to be deasserted.
6. Deassert the reset output to the Application Layer.

Related Information

Reset Sequence for Hard IP for PCI Express IP Core and Application Layer on page 8-2

Setting Up Simulation

Changing the simulation parameters reduces simulation time and provides greater visibility.

Changing Between Serial and PIPE Simulation

By default, the Altera testbench runs a serial simulation. You can change between serial and PIPE simulation by editing the top-level testbench file.

The `hip_ctrl_simu_mode_pipe` signal and `enable_pipe32_sim_hwtcl` parameter, specify serial or PIPE simulation. When both are set to 1'b0, the simulation runs in serial mode. When both are set to 1'b1, the simulation runs in PIPE mode. Complete the following steps to enable the 32-bit Gen3 PIPE simulation. These steps assume that the actual testbench is Gen3 x8 with an Avalon-ST 256-bit interface:

1. In the top-level testbench, which is **<*working_dir*>/<*variant*>/testbench/<*variant*>_*tb*simulation/<*variant*>tb.v**, change the signal, `hip_ctrl_simu_mode_pipe` to 1'b1 as shown:

```
pcie_de_gen3_x8_ast256 pcie_de_gen3_x8_ast256_inst (.
hip_ctrl_simu_mode_pipe ( 1'b1 ),
```

2. In the top-level HDL module for the Hard IP which is **<*working_dir*>/<*variant*>/testbench/<*variant*>_tb/simulation/submodules/<*variant*>.v** change the parameter `enable_pipe32_sim_hwtcl` parameter to 1'b1 as shown:

```
altpcie_<dev>_hip_ast_hwtcl #( .enable_pipe32_sim_hwtcl ( 1 ),
```

Using the PIPE Interface for Gen1 and Gen2 Variants

Running the simulation in PIPE mode reduces simulation time and provides greater visibility.

Complete the following steps to simulate using the PIPE interface:

1. Change to your simulation directory, **<*work_dir*>/<*variant*>/testbench/<*variant*>_tb/simulation**
2. Open <*variant*> **_tb.v**.
3. Search for the string, `serial_sim_hwtcl`. Set the value of this parameter to 0 if it is 1.
4. Save <*variant*>**_tb.v**.

Reducing Counter Values for Serial Simulations

You can accelerate simulation by reducing the value of counters whose default values are set for hardware, not simulation.

Complete the following steps to reduce counter values for simulation:

1. Open **<*work_dir*>/<*variant*>/testbench/<*variant*>_tb/simulation/submodules/altpcie_tbed_sv_hwtcl.v** .
2. Search for the string, `test_in`.
3. To reduce the value of several counters, set `test_in[0] = 1`.
4. Save **altpcietb_bfm_top_rp.v**.

Disable the Scrambler for Gen1 and Gen2 Simulations

The 128b/130b encoding scheme implemented by the scrambler applies a binary polynomial to the data stream to ensure enough data transitions between 0 and 1 to prevent clock drift. The data is decoded at the other end of the link by running the inverse polynomial.

Complete the following steps to disable the scrambler:

1. Open **<*work_dir*>/<*variant*>/testbench/<*variant*>_tb/simulation/submodules/altpcie_tbed_sv_hwtcl.v**.
2. Search for the string, `test_in`.
3. To disable the scrambler, set `test_in[2] = 1`.
4. Save **altpcie_tbed_sv_hwtcl.v**.

Changing between the Hard and Soft Reset Controller

The Hard IP for PCI Express includes both hard and soft reset control logic. By default, Gen1 ES and Gen1 and Gen2 production devices use the Hard Reset Controller. Gen2 and Gen3 ES devices and Gen3 production devices use the soft reset controller. For variants that use the hard reset controller, changing to the soft reset controller provides greater visibility.

Complete the following steps to change to the soft reset controller:

1. Open **<*work_dir*>/<*variant*>/testbench/<*variant*>_tb/simulation/submodules/<*variant*>.v**.
2. Search for the string, `hip_hard_reset_hwtcl`.
3. If `hip_hard_reset_hwtcl = 1`, the hard reset controller is active. Set `hip_hard_reset_hwtcl = 0` to change to the soft reset controller.
4. Save **variant.v**.

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 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.

There are two ways to eliminate this issue:

- You can perform a soft reset of the system to retain the FPGA programming while forcing the OS/BIOS to repeat its enumeration.
- You can use CvP to program the device.

Migrating PCIe Hard IP Qsys Design to Arria 10

A

December 2013

UG-01145_avmm

Subscribe Send Feedback

Introduction

This document helps you to migrate Qsys designs that use the Stratix V Hard IP for PCIe to the Arria 10 device family. This document uses the Stratix V device although the same process is applicable to all Altera 28 nm devices. With up-front planning that takes into account the differences in the Hard IP for PCIe, you can migrate your Stratix V PCI Express design to Arria 10.

Differences between Arria 10 and Stratix V Hard IP for PCIe in the Quartus II 13.1 A10 Release

The Arria 10 PCIe Hard IP uses a common GUI for Avalon-MM and Avalon-ST interfaces. The **Interface Type** parameter allows you to specify the interface.

The table below describes the difference between Arria 10 and Stratix V PCIe GUIs.

Table A-1: Differences between Arria 10 and Stratix V Hard IP for PCIe in the Quartus II 13.1 A10 Release

		Stratix V	Arria 10
System Settings	**PCIe Modes**	Gen1 x1 62.5 MHz	Must use Gen1 x1 125 MHz
		Gen3 x2 125 MHz 128 bits	Must use Gen3 x2 250 MHz 64 bits
	PCI Express Base Specification version	2.1 or 3.0	3.0 only
	Interface	Application Interface	Interface Type
	Reference Clock	100 MHz or 125 MHz	100 MHz
	Deprecated RX ST Byte Enable	Yes	Not supported
	Enable Configuration Bypass	Yes	Not supported
Base Address Registers, Base and Limit Register for Root Port, PCI Express/PCI Capabilities		Same	Same
Device Identification Register Defaults		Altera defaults	0
PHY Characteristics		ATX or CMU	ATX PLL only
		Common clock configuration	No Common clock configuration

Pinout Differences

Transceiver Reconfiguration Controller IP Core

The `reconfig_from_xcvr` and `reconfig_to_xcvr` buses are no longer present in the generated variation because the Arria 10 PCIe Hard IP no longer uses the Transceiver Reconfiguration Controller IP for calibration.

Local Management Interface

To reduce total pin count, the local management interface `lmi_din` and `lmi_dout` ports are 8 bits in Arria 10 variants. For Stratix V variants, these ports are 32 bits.

TX Credit Interface

The TX credit interface is time-division multiplexed for Arria 10.

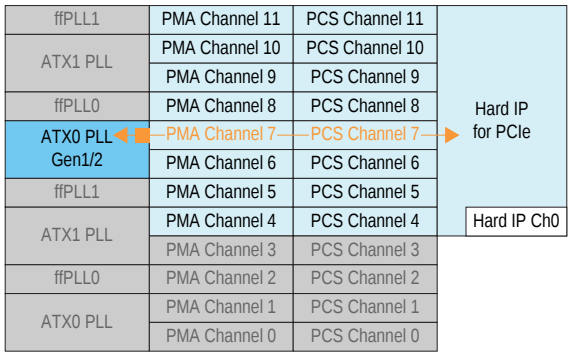
Table A-2: Differences between the TX Credit Interface for Arria 10 and Stratix V Devices

Arria 10			Stratix V		
Direction	Name	Width	Direction	Name	Width
output	tx_cred_data_fc	12	output	tx_cred_datafccp	12
output	tx_cred_hdr_fc	8	output	tx_cred_datafcnp	12
output	tx_cred_fc_hip_cons	6	output	tx_cred_datafccp	12
output	tx_cred_fc_infinite	6	output	tx_cred_hdrfccp	8
input	tx_cred_fc_sel	2	output	tx_cred_hdrfcnp	8
-	-	-	output	tx_cred_hdrfcp	8
-	-	-	output	tx_cred_fcchipcons	6
-	-	-	output	tx_cred_fcinfinite	6

Channel Placement

In Arria 10 device, PCI Express lane 4 always aligns with PCS channel 0 as shown in the Gen1 and Gen2 x8 channel placement diagram below:

Figure A-1: Channel Placement for the Arria 10 Hard IP for PCI Express Gen1 and Gen2 x8



Refer to the link below for comprehensive illustrations of channel placement.

Related Information

Channel Placement for PCIe In Arria 10 Devices on page 4-5

Qsys PCI Express Example Design

The design for this migration guide is based on the Gen3 x8 Stratix V PCIe Hard IP Qsys design example. The Qsys-generated top-level module is instantiated in a Verilog wrapper.

The Qsys design is available from a non-Arria 10 installation at **<*acds_install_path*>/ip/altera/altera_pcie/altera_pcie_hip_ast_ed/example_design/Stratix V/pcie_de_gen3_x8_ast256.qsys**.

The Qsys design instantiates the following five components:

- DUT—This is the user configured Stratix V Avalon-ST PCIe Hard IP. It is a Gen 3 x8 variation supporting two BARS.
- APPS—This is the Application Layer design. It instantiates a Chaining DMA engine connecting to the DUT Avalon-ST interface.
- pcie_reconfig_driver_0—This component contains the logic to reconfigure the transceiver for PCIe Gen3 speed negotiation.
- alt_xcvr_reconfig_0—This is the Transceiver Reconfiguration Controller IP core which dynamically reconfigures analog settings in the transceiver.
- clk_0—This component connects an external clock and reset to the pcie_reconfig_driver_0 and alt_xcvr_reconfig_0 components.

Quartus II A10 Software Device Migration flow

Follow these steps to open the design in Quartus II 13.1 A10 software release to begin migrating your design.

1. On the **File** menu, select **Open Project** to open the Stratix V design.
   a. Click **Yes** in the pop-up dialog box to overwrite the database with the A10 version of Quartus.
   b. Click **Yes** in the pop-up dialog box to remove all location assignments.
2. In the **Upgrade IP Component** dialog box, click the **IP Component** entity to be upgraded and click **Upgrade** to launch Qsys.

Update the Qsys Sub-System to Arria 10

Remove Transceiver Reconfiguration IP

1. On the **Tools** menu, select **Qsys .**

   Browse to the **.qsys** file and open it.
2. You can Ignore error messages and warning due to `device_family_hwtcl` and port differences.
3. In the **System Contents** pane, click on each of the following instances then click **Remove** to delete them:
   a. pcie_reconfig_driver_0
   b. alt_xcvr_reconfig_0
   c. clk_0

Upgrade PCIe Hard IP to A10 version

1. In the **Library** pane, double click on **Interface->PCI->Arria 10 Hard IP for PCI Express** to add an instance to your Qsys design
2. In the **Arria 10 Hard IP for PCI Express** GUI, configure the Arria10 instance to match the previous Stratix V instance. You can start another Qsys session to open the Stratix V instantiation for side by side comparison.

Complete the Arria 10 PCIe Hard IP Design

Complete the following steps to finish the Arria 10 design:

1. In the **Export** column, note the Stratix V exported ports and their names. Double click on each exported name to remove it.
2. In the **Export** column, double click on each of the Arria 10 ports that has the same name as the Stratix V previously exported port. Type in the same exported name that was used for the Stratix V Hard IP for PCI Express IP.
3. Right click on the **APPS** instance and select **Edit.**
4. Select **Arria 10** for **Targeted Device Family.**
5. Uncheck **Use deprecated RX Avalon-ST data byte enable port**.
6. If your Application Layer design uses the local management or the TX credit interfaces, you should update these interfaces to match port changes in the Arria 10 Hard IP for PCIe.
7. If your Application Layer design includes other blocks that target Stratix V, you need to retarget them to Arria 10.
8. Right click on each conduit interface of the Arria 10 Hard IP for PCIe and select target conduit interface from the pop-up connection menu.
9. Click on the Stratix V instance and select **Remove**.

Regenerate for Synthesis and Simulation

1. Click on **Generate** to build testbench and synthesis file for the design.
2. In the **Generate** dialog box, select **Standard, BFMs for Qsys interfaces** for **Create testbench Qsys system**.
3. Select **Verilog** or **VHDL** for **Create testbench simulation model**.
4. Select **Verilog** or **VHDL** for **Create HDL design files for synthesis**.

   Altera recommends that you use the same output directory structure and synthesis files as the Stratix V design to avoid manipulation of the project files.

5. Click on **Generate.**
6. Click on **Save** in the **Save changes** dialog box.

Simulate Your Arria 10 Design

To simulate the design, perform the following steps:

1. Start your simulation tool. This example uses the ModelSim software.
2. From the ModelSim transcript window, in the generated testbench directory type the following commands:

```
do msim_setup.tcl
ld_debug
run -all
```

Compile Your Arria 10 Design

Perform the following steps to compile your design in Quartus II A10:

1. On the **Assignment** menu, click **Device.**

Under the **Available** devices list, select the necessary Arria 10 device in the package, pin count and speed grade for your design.

2. On the **Assignment** menu, click **Pin Planner** to make I/O location assignments specific to your target Arria 10 device.
3. On the Processing menu, click **Start Compilation**.

Arria 10 PCIe Hard IP Migration Check List

You can use the following checklist to ensure that you have completed all steps for Stratix V to Arria 10 migration.

Table A-3: Arria 10 PCIe Hard IP Migration Check List

Item	Done	N/A	Arria 10 Hard IP
1			Review the Arria 10 Hard IP for PCI Express configuration to make sure that all intended features are configured and implemented
2			Review all sub-block to make sure that the targeted family is Arria 10
3			If the local management interface is used, revise your Application Layer logic to account for the width difference
4			If the TX credit interface is used, revise your Application Layer logic to account for port change
5			If the deprecated RX ST byte enable is used, revise your Application Layer logic
6			Remove the Transceiver Reconfiguration Controller IP and associated logic from your design
7			If CvP Update is used, it must be replaced by Partial Configuration
8			Review PCIe serial pinout (note that PCIe lane 0 aligns with PCS channel 4)
9			Review reset
10			Review timing constraints

Lane Initialization and Reversal

B

December 2013

UG-01145_avmm

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 10 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.

The following table lists the lane assignments for normal configuration.

Table B-1: 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

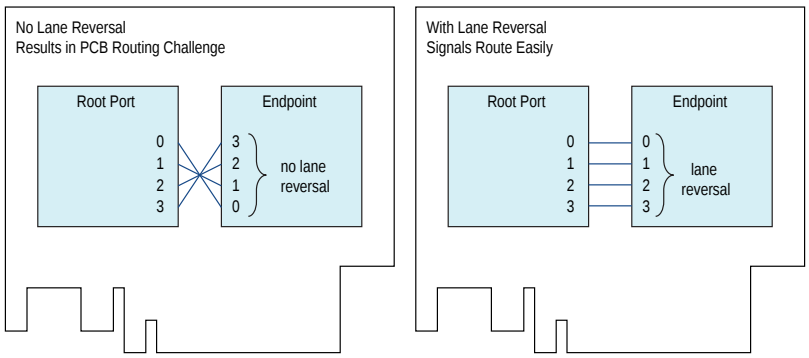
The following table lists the lane assignments with lane reversal.

Table B-2: 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

The following figure 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 B-1: Using Lane Reversal to Solve PCB Routing Problems



Transaction Layer Packet (TLP) Header Formats

C

December 2013

UG-01145_avmm

Subscribe Send Feedback

The following tables show the header format for TLPs without a data payload.

Figure C-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																															

Figure C-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																															

Figure C-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

Figure C-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						

Figure C-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																															

Figure C-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																															

Figure C-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	r2	r1	r0	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 .

Figure C-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																															

Figure C-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											
Byte 8	Requester ID																Tag								0	Lower Address						
Byte 12	Reserved																															

TLP Packet Formats with Data Payload

Figure C-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	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

Figure C-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

Figure C-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																															

Figure C-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																															

Figure C-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																															

Figure C-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																															

Figure C-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	r 2	r 1	r 0	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 D

December 2013

UG-01145_avmm Subscribe Send Feedback

Document Revision History

Date	Version	Changes
December 2013	13.1.0 A10	Initial release.

How to Contact Altera

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

Table D-1: Contact (1) Contact MethodAddress

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.

Related Information

- www.altera.com/support
- www.altera.com/training

- custrain@altera.com
- www.altera.com/literature
- nacomp@altera.com
- authorization@altera.com

Typographic Conventions

The following table shows the typographic conventions this document uses.

Table D-2: Visual CueMeaning

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.
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."

Visual Cue	Meaning
`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`).
r	An angled arrow instructs you to press the Enter key.
1., 2., 3., anda., 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.
1	The hand points to information that requires special attention.
h	The question mark directs you to a software help system with related information.
f	The feet direct you to another document or website with related information.
m	The multimedia icon directs you to a related multimedia presentation.
c	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
w	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.	
The feedback icon allows you to submit feedback to Altera about the document. Methods for collecting feedback vary as appropriate for each document.	

Visual Cue	Meaning
	The social media icons allow you to inform others about Altera documents. Methods for submitting information vary as appropriate for each medium.

Related Information

Email Subscription Management Center