Intel® Arria® 10 and Intel® Cyclone® 10 GX Avalon®-MM Interface for PCI Express* User Guide

Updated for Intel® Quartus® Prime Design Suite: **18.0**

Subscribe
Send Feedback

UG-01145_avmm | 2019.05.23
Latest document on the web: **PDF** | **HTML**

Contents

Send Feedback

1. Datasheet

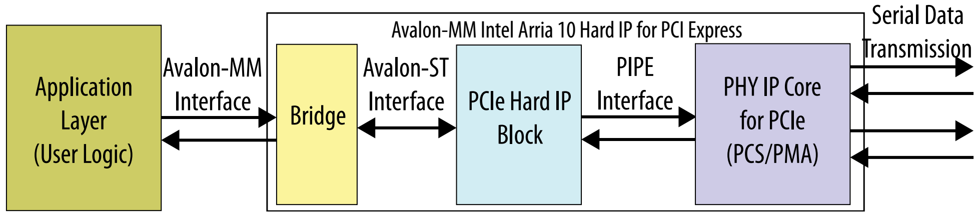
1.1. Intel® Arria® 10 or Intel® Cyclone® 10 GX Avalon-MM Interface for PCIe Datasheet

Intel® Arria® 10 and Intel Cyclone® 10 GX FPGAs include a configurable, hardened protocol stack for PCI Express* that is compliant with *PCI Express Base Specification 3.0* and *PCI Express Base Specification 2.0* respectively.

The Hard IP for PCI Express IP core using the Avalon® Memory-Mapped (Avalon-MM) interface removes some of the complexities associated with the PCIe* protocol. For example, it handles all of the Transaction Layer Packet (TLP) encoding and decoding. Consequently, you can complete your design more quickly. The Avalon-MM interface is implemented as a bridge in soft logic. It is available in Platform Designer.

Figure 1. Intel Arria 10 or Intel Cyclone 10 GX PCIe Variant with Avalon-MM Interface

The following figure shows the high-level modules and connecting interfaces for this variant.


Avalon-MM Intel Arria 10 Hard IP for PCI Express
Application Layer (User Logic)
Avalon-MM Interface
Bridge
Avalon-ST Interface
PCIe Hard IP Block
PIPE Interface
PHY IP Core for PCIe (PCS/PMA)
Serial Data Transmission

The following table shows the aggregate bandwidth of a PCI Express link for Gen1, Gen2, and Gen3 for 1, 2, 4, and 8 lanes. The protocol specifies 2.5 giga-transfers per second for Gen1, 5.0 giga-transfers per second for Gen2, and 8.0 giga-transfers per second for Gen3. This table provides bandwidths for a single transmit (TX) or receive (RX) channel. 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 about 1.5%.

Note: Intel Cyclone 10 GX support up to Gen2 x4 configurations.

Table 1. PCI Express Data Throughput

	Link Bandwidth in Gigabits Per Second (Gbps)			
	x1	x2	x4	x8
PCI Express Gen1 (2.5 Gbps)	2	4	8	16
PCI Express Gen2 (5.0 Gbps)	4	8	16	32
PCI Express Gen3 (8.0 Gbps)	7.87	15.75	31.5	63

Refer to *AN 456: PCI Express High Performance Reference Design* for more information about calculating bandwidth for the hard IP implementation of PCI Express in many Intel FPGAs.

Related Information

- Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM Interface for PCIe Solutions User Guide Archive on page 185
- Introduction to Intel FPGA IP Cores
  Provides general information about all Intel FPGA IP cores, including parameterizing, generating, upgrading, and simulating IP cores.
- Creating Version-Independent IP and Platform Designer Simulation Scripts
  Create simulation scripts that do not require manual updates for software or IP version upgrades.
- Project Management Best Practices
  Guidelines for efficient management and portability of your project and IP files.
- PCI Express Base Specification 3.0

1.2. Features

New features in the Quartus® Prime 18.0 software release:

- Added support for Intel Cyclone 10 GX devices for up to Gen2 x4 configurations.
- Added optional parameter to invert the RX polarity.

The Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express with the Avalon-MM interface supports the following features:

- Complete protocol stack including the Transaction, Data Link, and Physical Layers implemented as hard IP.
- Support for ×1, ×2, ×4, and ×8 configurations with Gen1, Gen2, or Gen3 lane rates for Intel Arria 10 Root Ports and Endpoints.
- Support for ×1, ×2, and ×4 configurations with Gen1 or Gen2 lane rates for Intel Cyclone 10 GX Root Ports and Endpoints.
- Dedicated 16 KB receive buffer.
- Optional support for Configuration via Protocol (CvP) using the PCIe link allowing the I/O and core bitstreams to be stored separately.
- Support for 32- or 64-bit addressing for the Avalon-MM interface to the Application Layer.
- Platform Designer design example demonstrating parameterization, design modules, and connectivity.
- Extended credit allocation settings to better optimize the RX buffer space based on application type.
- Optional end-to-end cyclic redundancy code (ECRC) generation and checking and advanced error reporting (AER) for high reliability applications.
- Easy to use:
  — Flexible configuration.
  — No license requirement.
  — Design examples to get started.

Table 2. Feature Comparison for all Hard IP for PCI Express IP Cores

The table compares the features of the three mainstream Hard IP for PCI Express IP Cores. Refer to the Intel Arria 10 *Avalon-ST Interface with SR-IOV PCIe Solutions User Guide* for the features of that variant.

Feature	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA
IP Core License	Free	Free	Free
Native Endpoint	Supported	Supported	Supported
Root port	Supported	Supported	Not supported
Gen1	×1, ×2, ×4, ×8	×1, ×2, ×4, ×8	x8
Gen2	×1, ×2, ×4, ×8	×1, ×2, ×4, ×8	×4, ×8
Gen3	×1, ×2, ×4, ×8	×1, ×2, ×4, x8 (1)	×2, ×4, ×8
64-bit Application Layer interface	Supported	Supported	Not supported
128-bit Application Layer interface	Supported	Supported	Supported
256-bit Application Layer interface	Supported	Supported	Supported
Maximum payload size	128, 256, 512, 1024, 2048 bytes	128, 256 bytes	128, 256 bytes
Number of tags supported for non-posted requests	32, 64, 128, or 256	8 for the 64-bit interface 16 for the 128-bit interface and 256-bit interface	16 or 256
Automatically handle out-of-order completions (transparent to the Application Layer)	Not supported	Not Supported	Not Supported
Automatically handle requests that cross 4 KB address boundary (transparent to the Application Layer)	Not supported	Supported	Supported
Polarity Inversion of PIPE interface signals	Supported	Supported	Supported
Number of MSI requests	1, 2, 4, 8, 16, or 32	1, 2, 4, 8, 16, or 32 (2)	1, 2, 4, 8, 16, or 32 (3)
MSI-X	Supported	Supported	Supported
Legacy interrupts	Supported	Supported	Supported
Expansion ROM	Supported	Not supported	Not supported
PCIe bifurcation	Not supported	Not supported	Not supported

Note: Intel Cyclone 10 GX devices support all the features in the table above, with the exception that they only support link width and speed combinations up to Gen2 x4.

(1) Gen3 x8 is supported in Root Port mode only.

(2) The Avalon-MM bridge can only send one MSI. However, the application logic can generate up to 32 MSI messages if needed since it has access to MSI information via the MSI-X interface.

(3) The Avalon-MM bridge can only send one MSI. However, the application logic can generate up to 32 MSI messages if needed since it has access to MSI information via the MSI-X interface.

Table 3. TLP Support Comparison for all Hard IP for PCI Express IP Cores

The table compares the TLP types that the Hard IP for PCI Express IP Cores variants can transmit. Each entry indicates whether this TLP type is supported (for transmit) by Endpoints (EP), Root Ports (RP), or both (EP/RP). For the Avalon-MM DMA interface, a software application programs a descriptor controller to specify DMA transfers between host and IP memory. The Read DMA Avalon-MM Master port and Write DMA Avalon-MM Master port send read and write TLPs, respectively. The optional TX Slave module supports single, non-bursting Memory Write TLPs to send status updates to the host.

TLP (Transmit Support)	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA
Memory Read Request (`Mrd`)	EP/RP	EP/RP	EP (Read DMA Avalon-MM Master)
Memory Read Lock Request (`MRdLk`)	EP/RP	Not supported	Not supported
Memory Write Request (`MWr`)	EP/RP	EP/RP	EP (Write DMA Avalon-MM Master) TX Slave (optional)
I/O Read Request (`IORd`)	EP/RP	EP/RP	Not supported
I/O Write Request (`IOWr`)	EP/RP	EP/RP	Not supported
Config Type 0 Read Request (`CfgRd0`)	RP	RP	Not supported
Config Type 0 Write Request (`CfgWr0`)	RP	RP	Not supported
Config Type 1 Read Request (`CfgRd1`)	RP	RP	Not supported
Config Type 1 Write Request (`CfgWr1`)	RP	RP	Not supported
Message Request (`Msg`)	EP/RP	Not supported	Not supported
Message Request with Data (`MsgD`)	EP/RP	Not supported	Not supported
Completion with Data (`CplD`)	EP/RP	EP/RP	EP (Read & Write DMA Avalon-MM Masters)
Completion-Locked (`CplLk`)	EP/RP	Not supported	Not supported
Completion Lock with Data (`CplDLk`)	EP/RP	Not supported	Not supported
Fetch and Add AtomicOp Request (`FetchAdd`)	EP	Not supported	Not supported

The *Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM Interface for PCIe Solutions User Guide* explains how to use this IP core and not the PCI Express protocol. Although there is inevitable overlap between these two purposes, use this document only in conjunction with an understanding of the *PCI Express Base Specification*.

Note: This release provides separate user guides for the different variants. The *Related Information* provides links to all versions.

Related Information

- Arria 10 and Cyclone 10 GX Avalon-MM DMA Interface for PCIe Solutions User Guide
  For the Avalon-MM interface and DMA functionality.

- Arria 10 and Cyclone 10 GX Avalon-ST Interface for PCIe Solutions User Guide
  For the Avalon-ST interface.
- Arria 10 Avalon-ST Interface with SR-IOV PCIe Solutions User Guide
  For the Avalon-ST interface with Single Root I/O Virtualization (SR-IOV).

1.3. Release Information

Table 4. Hard IP for PCI Express Release Information

Item	Description
Version	18.0
Release Date	May 2018
Ordering Codes	No ordering code is required
Product IDs	There are no encrypted files for the Intel Arria 10 or Intel Cyclone 10 GX 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	

Intel verifies that the current version of the Intel Quartus Prime software compiles the previous version of each IP core, if this IP core was included in the previous release. Intel reports any exceptions to this verification in the *Intel IP Release Notes* or clarifies them in the Intel Quartus Prime IP Update tool. Intel does not verify compilation with IP core versions older than the previous release.

Related Information

- Errata for the Arria 10 Hard IP for PCI Express IP Core in the Knowledge Base
- Errata for the Cyclone 10 GX Hard IP for PCI Express IP Core in the Knowledge Base
- Intel FPGA IP Release Notes
  Provides release notes for the current and past versions Intel FPGA IP cores.

1.4. Device Family Support

The following terms define device support levels for Intel FPGA IP cores:

- **Advance support**—the IP core is available for simulation and compilation for this device family. Timing models include initial engineering estimates of delays based on early post-layout information. The timing models are subject to change as silicon testing improves the correlation between the actual silicon and the timing models. You can use this IP core for system architecture and resource utilization studies, simulation, pinout, system latency assessments, basic timing assessments (pipeline budgeting), and I/O transfer strategy (data-path width, burst depth, I/O standards tradeoffs).
- **Preliminary support**—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.
- **Final support**—the IP core is verified with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

Table 5. Device Family Support

Device Family	Support Level
Intel Arria 10 or Intel Cyclone 10 GX	Final.
Other device families	Refer to the *Intel's PCI Express IP Solutions* web page for support information on other device families.

Related Information

PCI Express Solutions Web Page

1.5. Configurations

The Avalon-MM Intel Arria 10 Hard IP for PCI Express includes a full hard IP implementation of the PCI Express stack comprising the following layers:

- Physical (PHY), including:
  - Physical Media Attachment (PMA)
  - Physical Coding Sublayer (PCS)
- Media Access Control (MAC)
- Data Link Layer (DL)
- Transaction Layer (TL)

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

Figure 2. PCI Express Application with a Single Root Port and Endpoint

The following figure shows a PCI Express link between two Intel Arria 10 or Intel Cyclone 10 GX FPGAs.

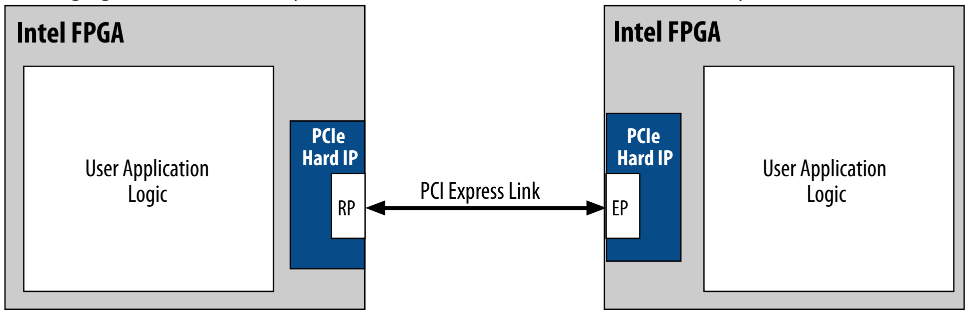
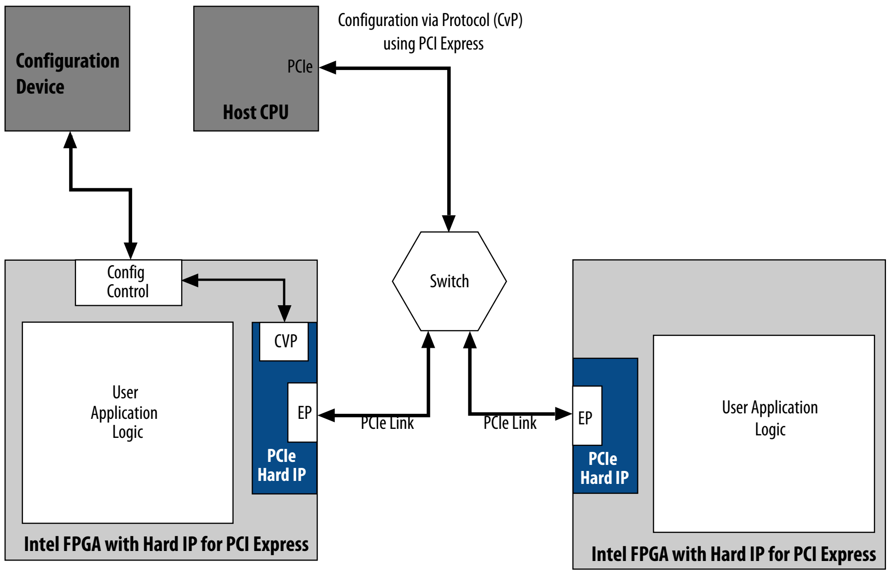


Figure 3. PCI Express Application Using Configuration via Protocol

The Intel Arria 10 design below includes the following components:

- Two Endpoints that connect to a PCIe switch.
- A host CPU that implements CvP using the PCI Express link connects through the switch.



Related Information

- Intel Arria 10 CvP Initialization and Partial Reconfiguration over PCI Express User Guide
- Intel Cyclone 10 GX CvP Initialization over PCI Express User Guide

1.6. Design Examples

Platform Designer example designs are available for the Avalon-MM Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express IP Core. You can download them from the `<install_dir>/ip/altera/altera_pcie/altera_pcie_a10_ed/example_design/a10` and `<install_dir>/ip/altera/altera_pcie/altera_pcie_a10_ed/example_design/c10` directories.

When you click the **Generate Example Design** button in the Parameter Editor, you are prompted to specify the example design location. After example design generation completes, this directory contains the customized example design that matches your parameter settings exactly; starting in the Quartus II software v15.0, this feature is available for most but not all IP core variations. If this feature is not available for your particular parameter settings, the Parameter Editor displays a warning.

Starting from the 18.0 release of the Intel Quartus Prime software, you can generate both Endpoint example designs and Root Port example designs.

1.7. IP Core Verification

To ensure compliance with the PCI Express specification, Intel performs extensive verification. The simulation environment uses multiple testbenches that consist of industry-standard bus functional models (BFMs) driving the PCI Express link interface. Intel performs the following tests in the simulation environment:

- Directed and pseudorandom stimuli test the Application Layer interface, Configuration Space, and all types and sizes of TLPs
- Error injection tests 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

Intel provides example designs that you can leverage to test your PCBs and complete compliance base board testing (CBB testing) at PCI-SIG, upon request.

1.7.1. Compatibility Testing Environment

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

1.8. Resource Utilization

Because the PCIe protocol stack is implemented in hardened logic, it uses no core device resources (no ALMs and no embedded memory).

The Avalon-MM soft logic bridge operates as a front end to the hardened protocol stack. The following table shows the typical device resource utilization for selected configurations using the current version of the Quartus Prime software. With the exception of M20K memory blocks, the numbers of ALMs and logic registers are rounded up to the nearest 50.

Table 6. Resource Utilization Avalon-MM Hard IP for PCI Express

Interface Width	ALMs	M20K Memory Blocks	Logic Registers
Avalon-MM Bridge			
64	1100	17	1500
128	1900	25	2900
Avalon-MM Interface–Completer Only			
64	650	8	1000
128	1400	12	2400
Avalon-MM–Completer Only Single Dword			
64	250	0	350

Related Information

Running the Fitter

1.9. Recommended Speed Grades

Table 7. Intel Cyclone 10 GX Recommended Speed Grades for All Avalon-MM Widths and Frequencies

Intel Cyclone 10 GX devices support only the Gen2 x4, 128-bit configuration in Avalon-MM mode.

Lane Rate	Link Width	Interface Width	Application Clock Frequency (MHz)	Recommended Speed Grades
Gen1	×1	64 bits	62.5, 125	–5, –6
	x2	64 bits	125	–5, –6
	x4	64 bits	125	–5, –6
Gen2	×1	64 bits	125	–5, –6
	x2	64 bits	125	–5, –6
	x4	64 bits	250	-5
	×4	128 bits	125	–5, –6

Table 8. Intel 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	×8	64 Bits	250	–1, –2
	×8	128 Bits	125	–1, –2, –3
	×1, x2, x4	64 Bits	125	–1, –2, –3
	×1	64 Bits	62.5	–1, –2, –3
Gen2	×8	128 bits	250	–1, –2
	×8	256 bits	125	–1, –2, -3
	×4	64 bits	250	–1, –2
	×4	128 bits	125	–1, –2, –3
	×1, x2	64 bits	125	–1, –2, –3
Gen3	×8	256 bits	250	–1, –2
	×4	128 bits	250	–1, –2
	×4	256 bits	125	–1, –2, –3
	×2	64 bits	250	–1, –2
	×2	128 bits	125	–1, –2, –3
	×1	64 bits	125	–1, –2, –3

Related Information

- Timing Closure and Optimization

- Intel FPGA Software Installation and Licensing
  Provides comprehensive information for installing and licensing Intel FPGA software.
- Running Synthesis

1.10. Creating a Design for PCI Express

Select the PCIe variant that best meets your design requirements.

- Is your design an Endpoint or Root Port?
- What Generation do you intend to implement?
- What link width do you intend to implement?
- What bandwidth does your application require?
- Does your design require Configuration via Protocol (CvP)?

Note: The following steps only provide a high-level overview of the design generation and simulation process. For more details, refer to the *Quick Start Guide* chapter.

1. Select parameters for that variant.
2. For Intel Arria 10 devices, you can use the new Example Design tab of the component GUI to generate a design that you specify. Then, you can simulate this example and also download it to an Intel Arria 10 FPGA Development Kit. Refer to the Intel Arria 10/Intel Cyclone 10 GX PCI Express IP Core Quick Start Guide for details.
3. For all devices, you can simulate using an Intel-provided example design. All static PCI Express example designs are available under `<install_dir>/ip/altera/altera_pcie/altera_pcie_<dev>_ed/example_design/<dev>`. Alternatively, create a simulation model and use your own custom or third-party BFM. The Platform Designer Generate menu generates simulation models. Intel supports ModelSim* - Intel FPGA Edition for all IP. The PCIe cores support the Aldec RivieraPro*, Cadence NCSim*, Mentor Graphics ModelSim, and Synopsys VCS* and VCS-MX* simulators.

   The Intel testbench and Root Port or Endpoint BFM provide a simple method to do basic testing of the Application Layer logic that interfaces to the variation. However, the testbench and Root Port BFM are not intended to be a substitute for a full verification environment. To thoroughly test your application, Intel suggests that you obtain commercially available PCI Express verification IP and tools, or do your own extensive hardware testing, or both.
4. Compile your design using the Quartus Prime software. If the versions of your design and the Quartus Prime software you are running do not match, regenerate your PCIe design.
5. Download your design to an Intel development board or your own PCB. Click on the *All Development Kits* link below for a list of Intel's development boards.
6. Test the hardware. You can use Intel's Signal Tap Logic Analyzer or a third-party protocol analyzer to observe behavior.
7. Substitute your Application Layer logic for the Application Layer logic in Intel's testbench. Then repeat Steps 3–6. In Intel's testbenches, the PCIe core is typically called the DUT (device under test). The Application Layer logic is typically called APPS.

Related Information

- Parameter Settings on page 25
- Quick Start Guide on page 17
- All Development Kits
- Intel Wiki PCI Express

For complete design examples and help creating new projects and specific functions, such as MSI or MSI-X related to PCI Express. Intel Applications engineers regularly update content and add new design examples. These examples help designers like you get more out of the Intel PCI Express IP core and may decrease your time-to-market. The design examples of the Intel Wiki page provide useful guidance for developing your own design. However, the content of the Intel Wiki is not guaranteed by Intel.

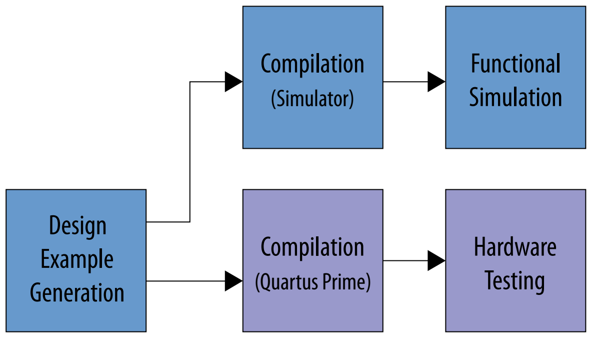
2. Quick Start Guide

The Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express IP core includes a programmed I/O (PIO) design example to help you understand usage. The PIO example transfers data from a host processor to a target device. It is appropriate for low-bandwidth applications. The design example includes an Avalon-ST to Avalon-MM Bridge. This component translates the TLPs received on the PCIe link to Avalon-MM memory reads and writes to the on-chip memory.

This design example automatically creates the files necessary to simulate and compile in the Quartus Prime software. You can download the compiled design to the Intel Arria 10 GX FPGA Development Kit. The design examples cover a wide range of parameters. However, the automatically generated design examples do not cover all possible parameterizations of the PCIe IP Core. If you select an unsupported parameter set, generations fails and provides an error message.

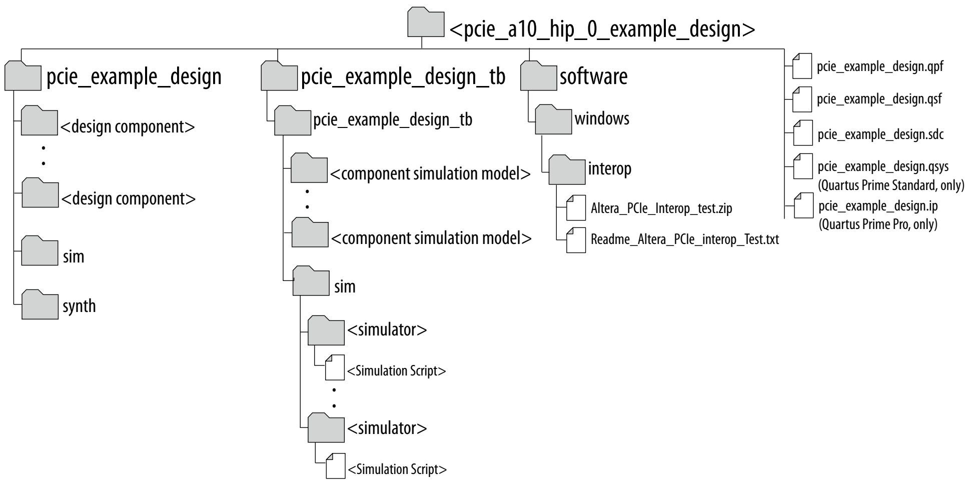
In addition, many static design examples for simulation are only available in the `<install_dir>/ip/altera/altera_pcie/altera_pcie_a10_ed/example_design/a10` and `<install_dir>/ip/altera/altera_pcie/altera_pcie_a10_ed/example_design/c10` directories.

Figure 4. Development Steps for the Design Example



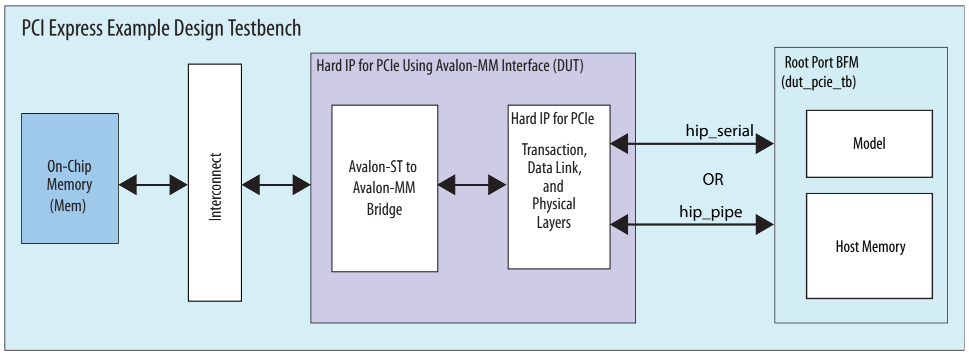
2.1. Directory Structure

Figure 5. Directory Structure for the Generated Design Example



2.2. Design Components for the Avalon-MM Endpoint

Figure 6. Block Diagram for the Platform Designer PIO Design Example Simulation Testbench


PCI Express Example Design Testbench
On-Chip Memory (Mem)
Interconnect
Hard IP for PCIe Using Avalon-MM Interface (DUT)
Avalon-ST to Avalon-MM Bridge
Hard IP for PCIe
Transaction, Data Link, and Physical Layers
hip_serial
OR
hip_pipe
Root Port BFM (dut_pcie_tb)
Model
Host Memory

2.3. Generating the Design

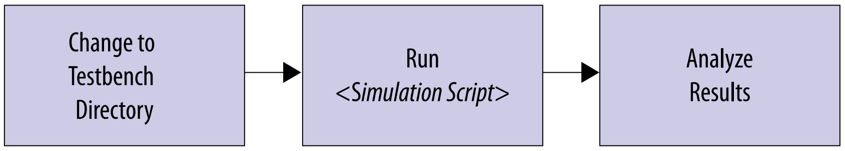
1. Launch Platform Designer.
   - If you have an existing .qsys file in your directory, the **Open System** dialog box appears. Click **New** to specify a Quartus Prime project name and custom IP variation name for your design. Then, click **Create**.
   - If not, a new project is automatically created. Save it before moving to the next step.
2. In the IP Catalog, locate and select **Intel Arria 10/Cyclone 10 Hard IP for PCI Express**. The parameter editor appears.
3. On the **IP Settings tabs**, specify the parameters for your IP variation.
4. In the Connections panel, make the following *dummy* connection: `rxm_bar0` to `txs` slave interface.

Platform Designer determines the size of the Avalon-MM BAR master from its connection to an Avalon-MM slave device. When you generate the example design, this connection is removed.

5. Remove the `clock_in` and `reset_in` components that were instantiated by default.
6. On the **Example Design** tab, the **PIO** design is available for your IP variation.
7. For **Example Design Files**, select the **Simulation** and **Synthesis** options.
8. For **Generated HDL Format**, only **Verilog** is available.
9. For **Target Development Kit**, select the **Intel Arria 10 GX FPGA Development Kit** option. Currently, there is no option to select an **Intel Cyclone 10 GX Development Kit** when generating an example design.
10. Click **Generate Example Design**. The software generates all files necessary to run simulations and hardware tests on the **Intel Arria 10 FPGA Development Kit**.

2.4. Simulating the Design

Figure 7. Procedure

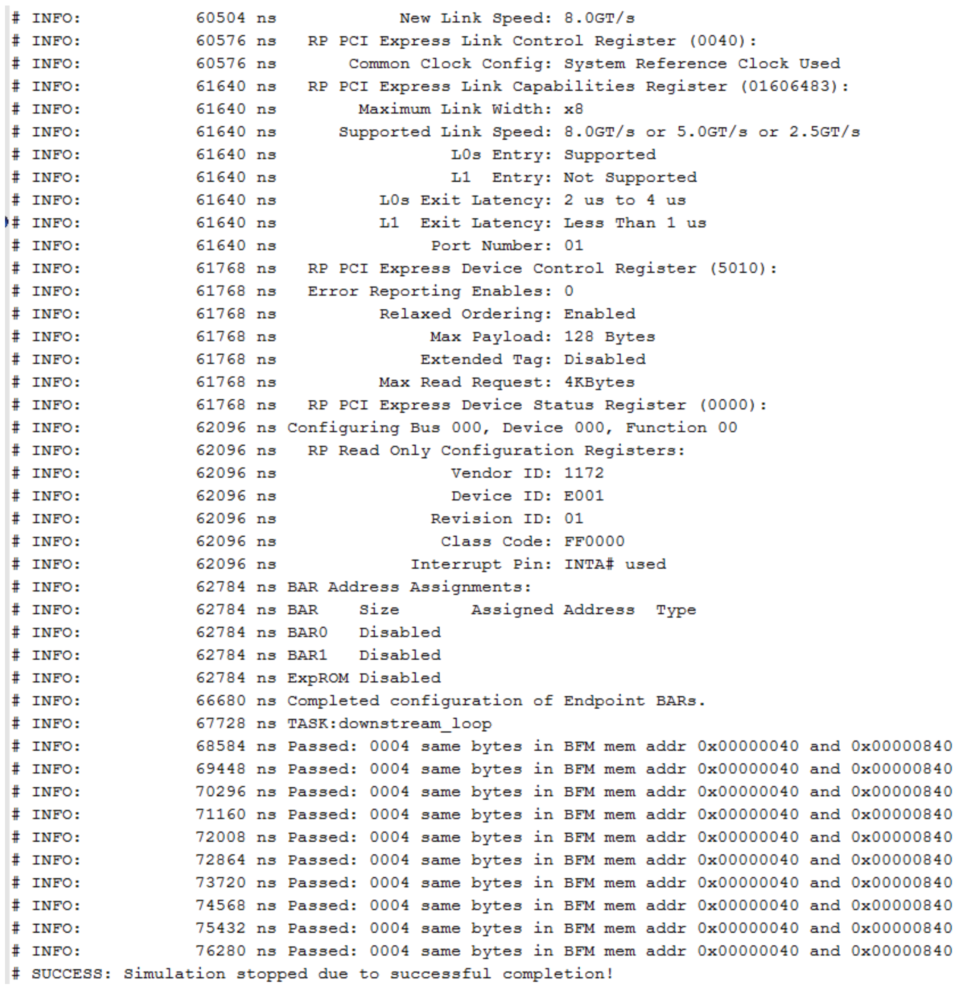


1. Change to the testbench simulation directory.
2. Run the simulation script for the simulator of your choice. Refer to the table below.
3. Analyze the results.

Table 9. Steps to Run Simulation

Simulator	Working Directory	Instructions
ModelSim	*<example_design>/* pcie_example_design_tb/ pcie_example_design_tb/sim/mentor/	1. Invoke vsim 2. `do msim_setup.tcl` 3. `ld_debug` 4. `run -all` 5. A successful simulation ends with the following message, "Simulation stopped due to successful completion!"
VCS	<example_design>/ pcie_example_design_tb/ pcie_example_design_tb/sim/ synopsys/vcs	1. sh vcs_setup.sh USER_DEFINED_SIM_OPTIONS="" 2. A successful simulation ends with the following message, "Simulation stopped due to successful completion!"
NCSim	*<example_design>/* pcie_example_design_tb/ pcie_example_design_tb/sim/cadence	1. sh ncsim_setup.sh USER_DEFINED_SIM_OPTIONS="" 2. A successful simulation ends with the following message, "Simulation stopped due to successful completion!"
Xcelium* Parallel Simulator	*<example_design>/* pcie_example_design_tb/ pcie_example_design_tb/sim/xcelium	1. sh xcelium_setup.sh USER_DEFINED_SIM_OPTIONS="" USER_DEFINED_ELAB_OPTIONS="-NOWARN\ CSINFI" 2. A successful simulation ends with the following message, "Simulation stopped due to successful completion!"

Figure 8. Partial Transcript from Successful Endpoint Avalon-ST PIO Simulation Testbench



```
# INFO:           60504 ns            New Link Speed: 8.0GT/s
# INFO:           60576 ns   RP PCI Express Link Control Register (0040):
# INFO:           60576 ns       Common Clock Config: System Reference Clock Used
# INFO:           61640 ns   RP PCI Express Link Capabilities Register (01606483):
# INFO:           61640 ns        Maximum Link Width: x8
# INFO:           61640 ns      Supported Link Speed: 8.0GT/s or 5.0GT/s or 2.5GT/s
# INFO:           61640 ns                 L0s Entry: Supported
# INFO:           61640 ns                 L1  Entry: Not Supported
# INFO:           61640 ns          L0s Exit Latency: 2 us to 4 us
# INFO:           61640 ns          L1  Exit Latency: Less Than 1 us
# INFO:           61640 ns               Port Number: 01
# INFO:           61768 ns   RP PCI Express Device Control Register (5010):
# INFO:           61768 ns   Error Reporting Enables: 0
# INFO:           61768 ns          Relaxed Ordering: Enabled
# INFO:           61768 ns               Max Payload: 128 Bytes
# INFO:           61768 ns              Extended Tag: Disabled
# INFO:           61768 ns          Max Read Request: 4KBytes
# INFO:           61768 ns   RP PCI Express Device Status Register (0000):
# INFO:           62096 ns Configuring Bus 000, Device 000, Function 00
# INFO:           62096 ns   RP Read Only Configuration Registers:
# INFO:           62096 ns                 Vendor ID: 1172
# INFO:           62096 ns                 Device ID: E001
# INFO:           62096 ns               Revision ID: 01
# INFO:           62096 ns                Class Code: FF0000
# INFO:           62096 ns             Interrupt Pin: INTA# used
# INFO:           62784 ns BAR Address Assignments:
# INFO:           62784 ns BAR    Size       Assigned Address  Type
# INFO:           62784 ns BAR0   Disabled
# INFO:           62784 ns BAR1   Disabled
# INFO:           62784 ns ExpROM Disabled
# INFO:           66680 ns Completed configuration of Endpoint BARs.
# INFO:           67728 ns TASK:downstream_loop
# INFO:           68584 ns Passed: 0004 same bytes in BFM mem addr 0x00000040 and 0x00000840
# INFO:           69448 ns Passed: 0004 same bytes in BFM mem addr 0x00000040 and 0x00000840
# INFO:           70296 ns Passed: 0004 same bytes in BFM mem addr 0x00000040 and 0x00000840
# INFO:           71160 ns Passed: 0004 same bytes in BFM mem addr 0x00000040 and 0x00000840
# INFO:           72008 ns Passed: 0004 same bytes in BFM mem addr 0x00000040 and 0x00000840
# INFO:           72864 ns Passed: 0004 same bytes in BFM mem addr 0x00000040 and 0x00000840
# INFO:           73720 ns Passed: 0004 same bytes in BFM mem addr 0x00000040 and 0x00000840
# INFO:           74568 ns Passed: 0004 same bytes in BFM mem addr 0x00000040 and 0x00000840
# INFO:           75432 ns Passed: 0004 same bytes in BFM mem addr 0x00000040 and 0x00000840
# INFO:           76280 ns Passed: 0004 same bytes in BFM mem addr 0x00000040 and 0x00000840
# SUCCESS: Simulation stopped due to successful completion!
```

2.5. Compiling and Testing the Design in Hardware

Figure 9. Procedure

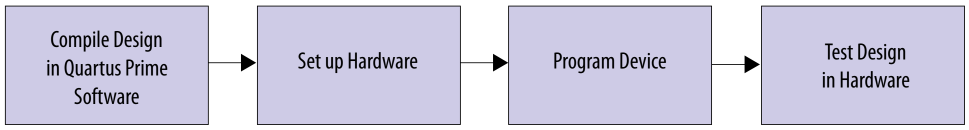
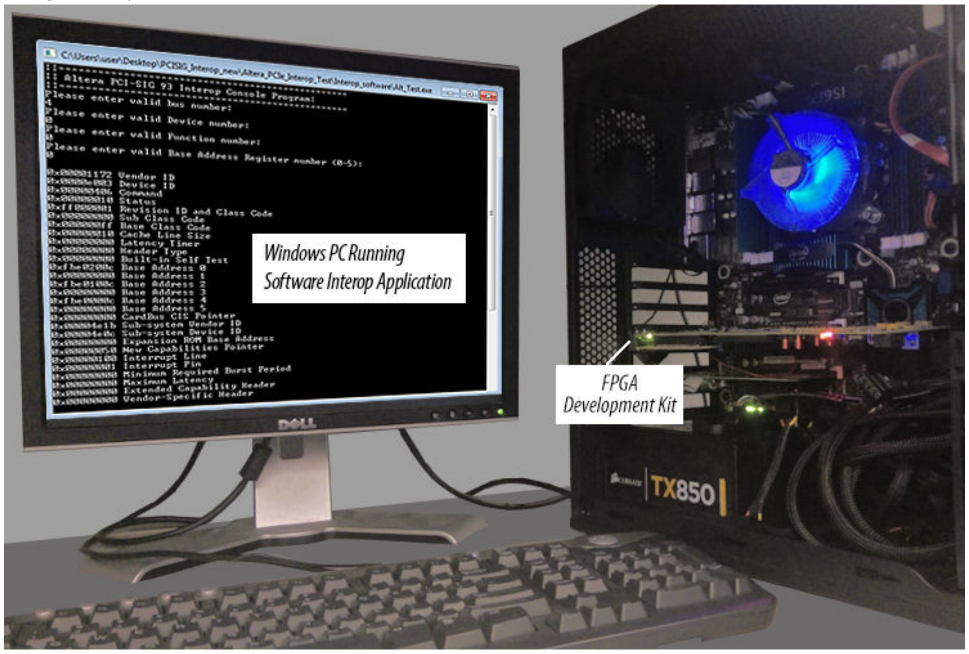


Figure 10. Software Application to Test the PCI Express Design Example on the Intel Arria 10 GX FPGA Development Kit

A software application running on a Windows PC performs the same hardware test for all of the PCI Express Design Examples.



```
C:\Users\user\Desktop\PCISIG_Interop_new\Altera_PCIe_Interop_Test\Interop_software\Alt_Test.exe
!!-----------------------------------------
!! Altera PCI-SIG 93 Interop Console Program!
!!-----------------------------------------
Please enter valid bus number:
4
Please enter valid Device number:
0
Please enter valid Function number:
0
Please enter valid Base Address Register number (0-5):
0

0x00001172 Vendor ID
0x0000e003 Device ID
0x00000406 Command
0x00000010 Status
0xff000001 Revision ID and Class Code
0x00000000 Sub Class Code
0x000000ff Base Class Code
0x00000010 Cache Line Size
0x00000000 Latency Timer
0x00000000 Header Type
0x00000000 Built-in Self Test
0xfbe0200c Base Address 0
0x00000000 Base Address 1
0xfbe0100c Base Address 2
0x00000000 Base Address 3
0xfbe0000c Base Address 4
0x00000000 Base Address 5
0x00000000 CardBus CIS Pointer
0x00004e1b Sub-system Vendor ID
0x00004e0c Sub-system Device ID
0x00000000 Expansion ROM Base Address
0x00000050 New Capabilities Pointer
0x00000100 Interrupt Line
0x00000001 Interrupt Pin
0x00000000 Minimum Required Burst Period
0x00000000 Maximum Latency
0x00000000 Extended Capability Header
0x00000000 Vendor-Specific Header
```

The software application to test the PCI Express Design Example on the Intel Arria 10 GX FPGA Development Kit is available on both 32- and 64-bit Windows platforms. This program performs the following tasks:

1. Prints the Configuration Space, lane rate, and lane width.
2. Writes 0x00000000 to the specified BAR at offset 0x00000000 to initialize the memory and read it back.
3. Writes 0xABCD1234 at offset 0x00000000 of the specified BAR. Reads it back and compares.

If successful, the test program displays the message 'PASSED'

Follow these steps to compile the design example in the Quartus Prime software:

1. Launch the Quartus Prime software and open the `pcie_example_design.qpf` file for the example design created above.
2. On the **Processing** > menu, select **Start Compilation**.

   The timing constraints for the design example and the design components are automatically loaded during compilation.

Follow these steps to test the design example in hardware:

1. In the *<example_design>*`/software/windows/interop` directory, unzip `Altera_PCIe_Interop_Test.zip`.

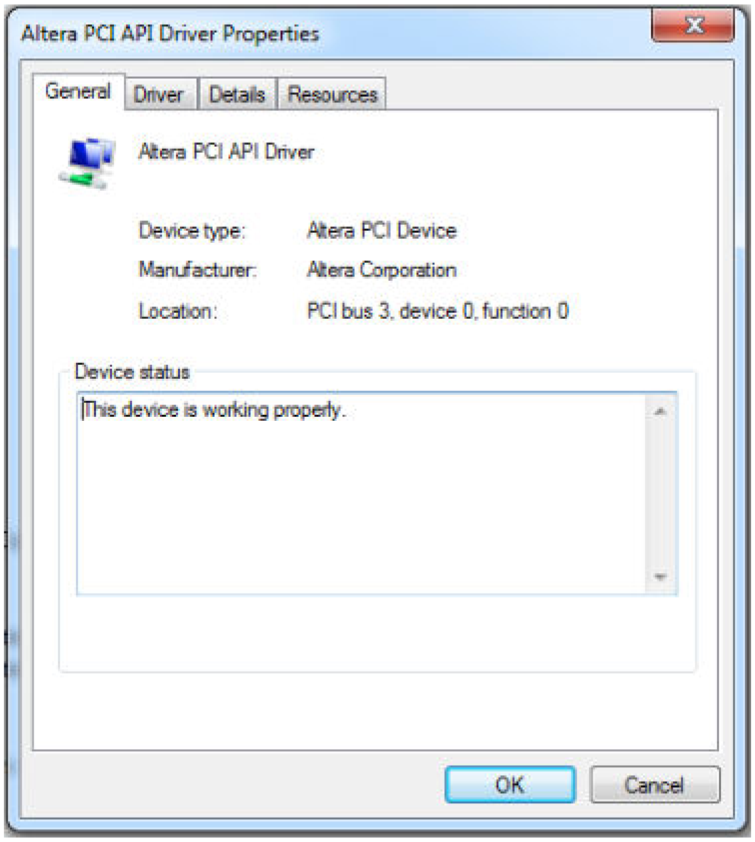
Note: You can also refer to `readme_Altera_PCIe_interop_Test.txt` file in this same directory for instructions on running the hardware test.

2. Install the Intel FPGA Windows Demo Driver for PCIe on the Windows host machine, using **altera_pcie_win_driver.inf**.

   Note: If you modified the default Vendor ID (0x1172) or Device ID (0x0000) specified in the component parameter editor GUI, you must also modify them in **altera_pcie_win_driver.inf**.

   a. In the <*`example_design`*> directory, launch the Quartus Prime software and compile the design (**Processing** > **Start Compilation**).
   b. Connect the development board to the host computer.
   c. Configure the FPGA on the development board using the generated `.sof` file (**Tools** > **Programmer**).
   d. Open the Windows Device Manager and scan for hardware changes.
   e. Select the Intel FPGA listed as an unknown PCI device and point to the appropriate 32- or 64-bit driver (**altera_pice_win_driver.inf**) in the **Windows_driver** directory.
   f. After the driver loads successfully, a new device named **Altera PCI API Device** appears in the Windows Device Manager.
   g. Determine the bus, device, and function number for the **Altera PCI API Device** listed in the Windows Device Manager.
      i. Expand the tab, **Altera PCI API Driver** under the devices.
      ii. Right click on **Altera PCI API Device** and select **Properties**.
      iii. Note the bus, device, and function number for the device. The following figure shows one example.

Figure 11. Determining the Bus, Device, and Function Number for New PCIe Device



Device type: Altera PCI Device
Manufacturer: Altera Corporation
Location: PCI bus 3, device 0, function 0

This device is working properly.

3. In the <*example_design*>`/software/windows/interop/Altera_PCIe_Interop_Test/Interop_software` directory, click `Alt_Test.exe`.
4. When prompted, type the bus, device, and function numbers and select the BAR number (0-5) you specified when parameterizing the IP core.

   Note: The bus, device, and function numbers for your hardware setup may be different.
5. The test displays the message, PASSED, if the test is successful.

Note: For more details on additional design implementation steps such as making pin assignments and adding timing constraints, refer to the *Design Implementation* chapter.

Related Information

- Intel Arria 10 Development Kit Conduit Interface on page 62
- Intel Arria 10 GX FPGA Development Kit

3. Parameter Settings

3.1. Parameters

This chapter provides a reference for all the parameters of the Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express IP core.

Table 10. Design Environment Parameter

Starting in Intel Quartus Prime 18.0, there is a new parameter **Design Environment** in the parameters editor window.

Parameter	Value	Description
Design Environment	**Standalone** **System**	Identifies the environment that the IP is in. • The **Standalone** environment refers to the IP being in a standalone state where all its interfaces are exported. • The **System** environment refers to the IP being instantiated in a Platform Designer system.

Table 11. System Settings

Parameter	Value	Description
Application Interface Type	**Avalon-ST** **Avalon-MM** **Avalon-MM with DMA** **Avalon-ST with SR-IOV**	Selects the interface to the Application Layer. *Note:* When the **Design Environment** parameter is set to **System**, all four **Application Interface Types** are available. However, when **Design Environment** is set to **Standalone**, only **Avalon-ST** and **Avalon-ST with SR-IOV** are available.
Hard IP mode	**Gen3x8, Interface: 256-bit, 250 MHz** **Gen3x4, Interface: 256-bit, 125 MHz** **Gen3x4, Interface: 128-bit, 250 MHz** **Gen3x2, Interface: 128-bit, 125 MHz** **Gen3x2, Interface: 64-bit, 250 MHz** **Gen3x1, Interface: 64-bit, 125 MHz** **Gen2x8, Interface: 256-bit, 125 MHz** **Gen2x8, Interface: 128-bit, 250 MHz** **Gen2x4, Interface: 128-bit, 125 MHz** **Gen2x2, Interface: 64-bit, 125 MHz** **Gen2x4, Interface: 64-bit, 250 MHz** **Gen2x1, Interface: 64-bit, 125 MHz** **Gen1x8, Interface: 128-bit, 125 MHz** **Gen1x8, Interface: 64-bit, 250 MHz** **Gen1x4, Interface: 64-bit, 125 MHz** **Gen1x2, Interface: 64-bit, 125 MHz** **Gen1x1, Interface: 64-bit, 125 MHz** **Gen1x1, Interface: 64-bit, 62.5 MHz**	Selects the following elements: • The lane data rate. Gen1, Gen2, and Gen3 are supported • The width of the data interface between the hard IP Transaction Layer and the Application Layer implemented in the FPGA fabric • The Application Layer interface frequency Intel Cyclone 10 GX devices support up to Gen2 x4 configurations.
Port type	**Native Endpoint**	Specifies the port type.

continued...

Parameter	Value	Description
	Root Port	The Endpoint stores parameters in the Type 0 Configuration Space. The Root Port stores parameters in the Type 1 Configuration Space. You can enable the Root Port in the current release. Root Port mode only supports the Avalon-MM interface type, and it only supports basic simulation and compilation. However, the Root Port mode is not fully verified.
RX Buffer credit allocation - performance for received requests	**Minimum** **Low** **Balanced**	Determines the allocation of posted header credits, posted data credits, non-posted header credits, completion header credits, and completion data credits in the 16 KB RX buffer. The settings allow you to adjust the credit allocation to optimize your system. The credit allocation for the selected setting displays in the **Message** pane. The **Message** pane dynamically updates the number of credits for Posted, Non-Posted Headers and Data, and Completion Headers and Data as you change this selection. Refer to the *Throughput Optimization* chapter for more information about optimizing your design. Refer to the *RX Buffer Allocation Selections Available by Interface Type* below for the availability of these settings by interface type. **Minimum**—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**—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**—configures 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.
RX Buffer completion credits	**Header credits** **Data credits**	Displays the number of completion credits in the 16 KB RX buffer resulting from the credit allocation parameter. Each header credit is 16 bytes. Each data credit is 20 bytes.

Related Information

Throughput Optimization on page 109

3.2. Avalon-MM Settings

Table 12. Avalon-MM Settings

Parameter	Value	Description
Avalon-MM address width	**32-bit**	Specifies the address width for Avalon-MM RX master ports that access Avalon-MM slaves in the Avalon address domain.

continued...

Send Feedback

Parameter	Value	Description
	64-bit	When you select **Enable Avalon-MM DMA** or **Enable non-bursting Avalon-MM slave interface with individual byte access (TXS)**, this value must be set to **64**.
Enable completer-only Endpoint	**On/Off**	In **completer-only** mode, the Hard IP can receive requests, but cannot initiate upstream requests. However, it can transmit completion packets on the PCI Express TX link. This mode removes the Avalon-MM TX slave port and thereby reduces logic utilization.
Enable completer-only Endpoint with 4-byte payload	**On/Off**	This is a non-pipelined version of **Completer Only** mode. At any time, only a single request can be outstanding. **Single DWORD completer** uses fewer resources than **Completer Only**. This variant is targeted for systems that require simple read and write register accesses from a host CPU. If you select this option, the width of the data for RXM BAR masters is always 32 bits, regardless of the **Avalon-MM** width. For the Avalon-MM interface with DMA, this value must be **Off**.
Enable control register access (CRA) Avalon-MM slave port	**On/Off**	Allows read and write access to bridge registers from the interconnect fabric using a specialized slave port. This option is required for **Requester/Completer** variants and optional for **Completer Only** variants. Enabling this option allows read and write access to bridge registers, except in the Completer-Only single DWORD variations.
Export MSI/MSI-X conduit interfaces	**On/Off**	When you turn this option **On**, the core exports top-level MSI and MSI-X interfaces that you can use to implement a Custom Interrupt Handler for MSI and MSI-X interrupts. For more information about the Custom Interrupt Handler, refer to *Interrupts for End Points Using the Avalon-MM Interface with Multiple MSI/MSI-X Support*. If you turn this option **Off**, the core handles interrupts internally.
Enable PCIe interrupt at power-on	**On/Off**	When you turn this option **On**, the Avalon-MM Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express the interrupt register is enabled at power-up. Turning off this option disables the interrupt register at power-up. The setting does not affect run-time configuration of the interrupt enable register. For the Avalon-MM interface with DMA, this value must be **Off**.
Enable hard IP status bus when using the Avalon-MM interface	**On/Off**	When you turn this option **On**, your top-level variant includes signals that are useful for debugging, including link training and status, and error signals. The following signals are included in the top-level variant: • Link status signals • ECC error signals • LTSSM signals • Configuration parity error signal
Address width of accessible PCIe memory space	**20-64**	Specifies the number of bits necessary to access the PCIe address space.

3.3. Base Address Register (BAR) Settings

You can configure up to six 32-bit BARs or three 64-bit BARs.

Table 13. BAR Registers

Parameter	Value	Description
Type	**Disabled** **64-bit prefetchable memory** **32-bit non-prefetchable memory** **32-bit prefetchable memory**	Defining memory as prefetchable allows data in the region to be fetched ahead anticipating that the requestor may require more data from the same region than was originally requested. If you specify that a memory is prefetchable, it must have the following 2 attributes:

continued...

Parameter	Value	Description
	I/O address space	• Reads do not have side effects • Write merging is allowed The **32-bit prefetchable memory** and **I/O address space** BARs are only available for the **Legacy Endpoint**.
Size	Not configurable	Specifies the memory size calculated from other parameters you enter.

3.4. Device Identification Registers

Table 14. Device ID 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. Refer to *Type 0 Configuration Space Registers* for the layout of the Device Identification registers.

Register Name	Range	Default Value	Description
Vendor ID	16 bits	0x00001172	Sets the read-only value of the `Vendor ID` register. This parameter cannot 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. This register is only valid in the Type 0 (Endpoint) Configuration Space. 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. The 24-bit `Class Code` register is further divided into three 8-bit fields: `Base Class Code`, `Sub-Class Code` and `Programming Interface`. For more details on these fields, refer to the *PCI Express Base Specification*. 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*. This value is assigned by PCI-SIG to the device manufacturer. This register is only valid in the Type 0 (Endpoint) Configuration Space. Address offset: 0x02C.
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

3.5. 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 indicates the parameter address.

3.5.1. Device Capabilities

Table 15. 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**.**
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 the 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 specify the range: • None—Completion timeout programming is not supported • 0001 Range A • 0010 Range B • 0011 Ranges A and B • 0110 Ranges B and C • 0111 Ranges A, B, and C • 1110 Ranges B, C and D • 1111 Ranges A, B, C, and D All other values are reserved. Intel 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.

3.5.2. Error Reporting

Table 16. Error Reporting

Parameter	Value	Default Value	Description
Enable Advanced Error Reporting (AER)	**On/Off**	Off	When **On**, enables the Advanced Error Reporting (AER) capability.
Enable ECRC checking	**On/Off**	Off	When **On**, enables ECRC checking. Sets the read-only value of the ECRC check capable bit in the `Advanced Error Capabilities and Control Register`. This parameter requires you to enable the AER capability.

continued...

Parameter	Value	Default Value	Description
Enable 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.
Enable ECRC forwarding on the Avalon-ST interface	**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 or Avalon-MM DMA interfaces.
Track RX completion buffer overflow on the Avalon-ST interface	**On/Off**	Off	When **On**, the core includes the `rxfc_cplbuf_ovf` output status signal to track the RX posted completion buffer overflow status. Not applicable for Avalon-MM or Avalon-MM DMA interfaces.

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.

3.5.3. Link Capabilities

Table 17. Link Capabilities

Parameter	Value	Description
Link port number (Root Port only)	**0x01**	Sets the read-only value of the port number field in the `Link Capabilities` register. This parameter is for Root Ports only. It should not be changed.
Data link layer active reporting (Root Port only)	**On/Off**	Turn **On** this parameter for a Root Port, if the attached Endpoint supports the optional capability of reporting the DL_Active state of the Data Link Control and Management State Machine. For a hot-plug capable Endpoint (as indicated by the `Hot Plug Capable` field of the `Slot Capabilities` register), this parameter must be turned **On**. For Root Port components that do not support this optional capability, turn **Off** this option. Not applicable for Avalon-MM or Avalon-MM DMA interfaces.
Surprise down reporting (Root Port only)	**On/Off**	When your turn this option **On**, an Endpoint supports the optional capability of detecting and reporting the surprise down error condition. The error condition is read from the Root Port. Not applicable for Avalon-MM or Avalon-MM DMA interfaces.
Slot clock configuration	**On/Off**	When you turn this option **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. This parameter sets the Slot Clock Configuration bit (bit 12) in the `PCI Express Link Status` register.

3.5.4. MSI and MSI-X Capabilities

Table 18. 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, Address: 0x050[31:16].
	MSI-X Capabilities	
Implement MSI-X	**On/Off**	When **On**, adds the MSI-X functionality.
	Bit Range	
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 in the MSI-X Capability Structure. Legal range is 0–2047 (2^{11}). Address offset: 0x068[26:16]
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 64-bit qword-aligned offset. This field is read-only.
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.
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 in the MSI-X Capability Structure. (4)
Pending 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 in the MSI-X Capability Structure. Legal range is 0–5.

3.5.5. Slot Capabilities

Table 19. Slot Capabilities

Parameter	Value	Description
Use Slot register	**On/Off**	This parameter is only supported in Root Port mode. 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. Defines the characteristics of the slot. You turn on this option by selecting **Enable slot capability**. Refer to the figure below for bit definitions.
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

continued...

(4) 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.

Parameter	Value	Description
		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.

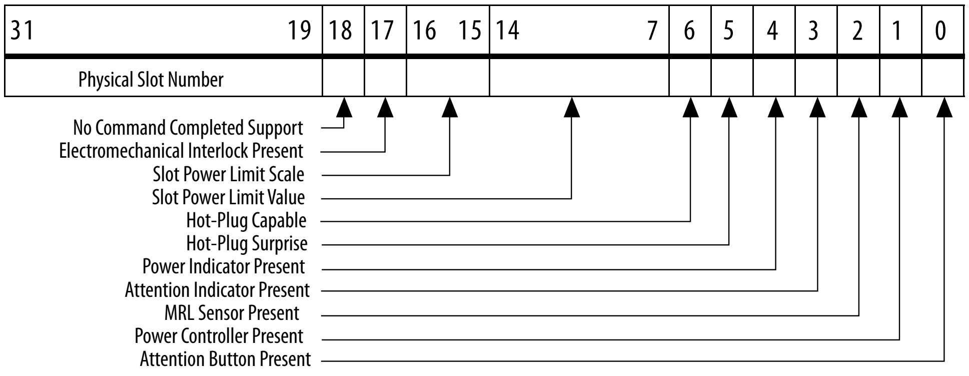


Figure 12. Slot Capability

3.5.6. Power Management

Table 20. Power Management Parameters

Parameter	Value	Description
Endpoint L0s acceptable latency	**Maximum of 64 ns** **Maximum of 128 ns** **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 a safe 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** **Maximum of 64 ns** **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, a switched system may include 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.

continued...

Parameter	Value	Description
		The default value of this parameter is 1 µs. This is a safe setting for most designs.

The Intel Stratix® 10 Avalon-ST Hard IP for PCI Express and Intel Stratix 10 Avalon-MM Hard IP for PCI Express do not support the L1 or L2 low power states. If the link ever gets into these states, performing a reset (by asserting `pin_perst`, for example) allows the IP core to exit the low power state and the system to recover.

These IP cores also do not support the in-band beacon or sideband WAKE# signal, which are mechanisms to signal a wake-up event to the upstream device.

3.6. Configuration, Debug, and Extension Options

Table 21. System Settings for PCI Express

Parameter	Value	Description
Enable configuration via Protocol (CvP)	**On/Off**	When **On**, the Quartus Prime 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 supported for Intel Cyclone 10 GX devices from the Intel Quartus Prime release 17.1.1 onwards.
Enable dynamic reconfiguration of PCIe read-only registers	**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 transceiver dynamic reconfiguration	**On/Off**	When on, creates an Avalon-MM slave interface that software can drive to update transceiver registers.
Enable Altera Debug Master Endpoint (ADME)	**On/Off**	When **On**, an embedded Altera Debug Master Endpoint connects internally to the Avalon-MM slave interface for dynamic reconfiguration. The ADME can access the reconfiguration space of the transceiver. It uses JTAG via the System Console to run tests and debug functions.
Enable Intel Arria 10 FPGA Development Kit connection	**On/Off**	When **On**, add control and status conduit interface to the top level variant, to be connected a PCIe Development Kit component.

Related Information

Configuration over Protocol (CvP) on page 113

3.7. Vendor Specific Extended Capability (VSEC)

Table 22. VSEC

Parameter	Value	Description
Vendor Specific Extended Capability (VSEC) ID:	0x00001172	Sets the read-only value of the 16-bit User ID register from the Vendor Specific Extended Capability.
Vendor Specific Extended Capability (VSEC) Revision:	0x00000000	Sets the read-only value of the 4-bit VSEC Revision register from the Vendor Specific Extended Capability.
User Device or Board Type ID register from the Vendor Specific Extended Capability:	0x00000000	Sets the read-only value of the 16-bit Device or Board Type ID register from the Vendor Specific Extended Capability.

3.8. PHY Characteristics

Table 23. PHY Characteristics

Parameter	Value	Description
Gen2 TX de-emphasis	**3.5dB** **6dB**	Specifies the transmit de-emphasis for Gen2. Intel recommends the following settings: • 3.5dB: Short PCB traces • 6.0dB: Long PCB traces.
Requested equalization far-end TX preset	**Preset0-Preset9**	Specifies the requested TX preset for Phase 2 and 3 far-end transmitter. The default value **Preset8** provides the best signal quality for most designs.
Enable soft DFE controller IP	**On** **Off**	When **On**, the PCIe Hard IP core includes a decision feedback equalization (DFE) soft controller in the FPGA fabric to improve the bit error rate (BER) margin. The default for this option is **Off** because the DFE controller is typically not required. However, short reflective links may benefit from this soft DFE controller IP. This parameter is available only for Gen3 mode. It is not supported when CvP or autonomous modes are enabled.
Enable RX-polarity inversion in soft logic	**On** **Off**	This parameter mitigates the following RX-polarity inversion problem. When the Intel Arria 10 or Intel Cyclone 10 GX Hard IP core receives TS2 training sequences during the Polling.Config state, when you have not enabled this parameter, automatic lane polarity inversion is not guaranteed. The link may train to a smaller than expected link width or may not train successfully. This problem can affect configurations with any PCIe speed and width. When you include this parameter, polarity inversion is available for all configurations except Gen1 x1. This fix does not support CvP or autonomous mode.

3.9. Example Designs

Table 24. Example Designs

Parameter	Value	Description
Available Example Designs	**DMA** **PIO**	When you select the **DMA** option, the generated example design includes a direct memory access application. This application includes upstream and downstream transactions.

continued...

Send Feedback

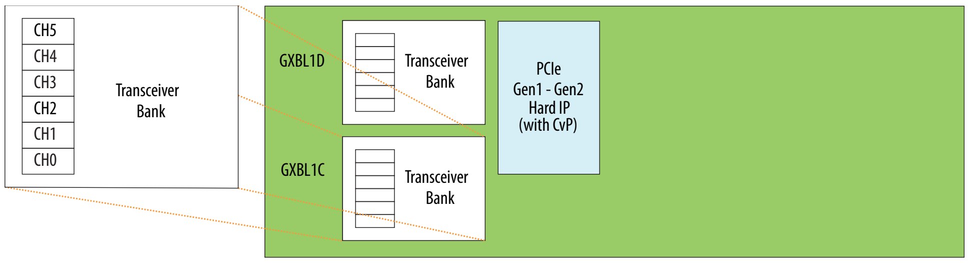
Parameter	Value	Description
		When you select the PIO option, the generated design includes a target application including only downstream transactions.
Simulation	**On/Off**	When **On**, the generated output includes a simulation model.
Synthesis	**On/Off**	When **On**, the generated output includes a synthesis model.
Generated HDL format	**Verilog/VHDL**	Verilog HDL and VHDL are supported
Select Board	**Intel Arria 10 FPGA GX Development Kit** **Intel Arria 10 FPGA GX Development Kit ES2** **None**	Specifies the Intel Arria 10 development kit. Select **None** to download to a custom board. *Note:* Currently, you cannot target an Intel Cyclone 10 GX Development Kit when generating an example design.

4. Physical Layout

4.1. Hard IP Block Placement In Intel Cyclone 10 GX Devices

Intel Cyclone 10 GX devices include a single hard IP blocks for PCI Express. This hard IP block includes the CvP functionality for flip chip packages.

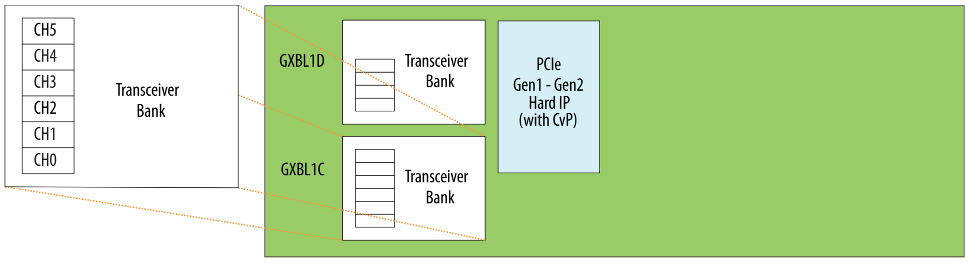
Figure 13. Intel Cyclone 10 GX Devices with 12 Transceiver Channels and One PCIe Hard IP Block



Legend:

🟦	PCIe Gen1 - Gen2 Hard IP blocks with Configuration via Protocol (CvP) capabilities.
🟩	Cyclone 10 GX device with 12 transceiver channels and one PCIe Hard IP block.

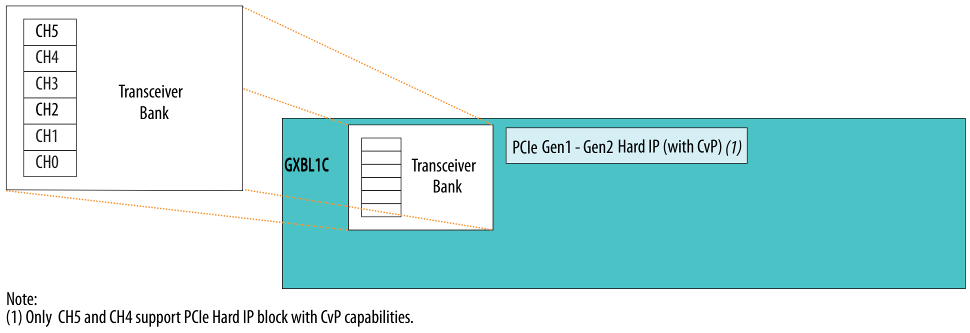
Figure 14. Intel Cyclone 10 GX Devices with 10 Transceiver Channels and One PCIe Hard IP Block



Legend:

🟦	PCIe Gen1 - Gen2 Hard IP blocks with Configuration via Protocol (CvP) capabilities.
🟩	Cyclone 10 GX device with 10 transceiver channels and one PCIe Hard IP block.

Figure 15. Intel Cyclone 10 GX Devices with 6 Transceiver Channels and One PCIe Hard IP Block


CH5
CH4
CH3
CH2
CH1
CH0
Transceiver Bank
GXBL1C
Transceiver Bank
PCIe Gen1 - Gen2 Hard IP (with CvP) (1)
Note:
(1) Only CH5 and CH4 support PCIe Hard IP block with CvP capabilities.

Legend:

⬜	PCIe Gen1 - Gen2 Hard IP block with Configuration via Protocol (CvP) capabilities.
🟦	Cyclone 10 GX device with six transceiver channels and one PCIe Hard IP block.

Refer to the *Intel Cyclone 10 GX Device Transceiver Layout* in the *Intel Cyclone 10 GX Transceiver PHY User Guide* for comprehensive figures for Intel Cyclone 10 GX devices.

Related Information

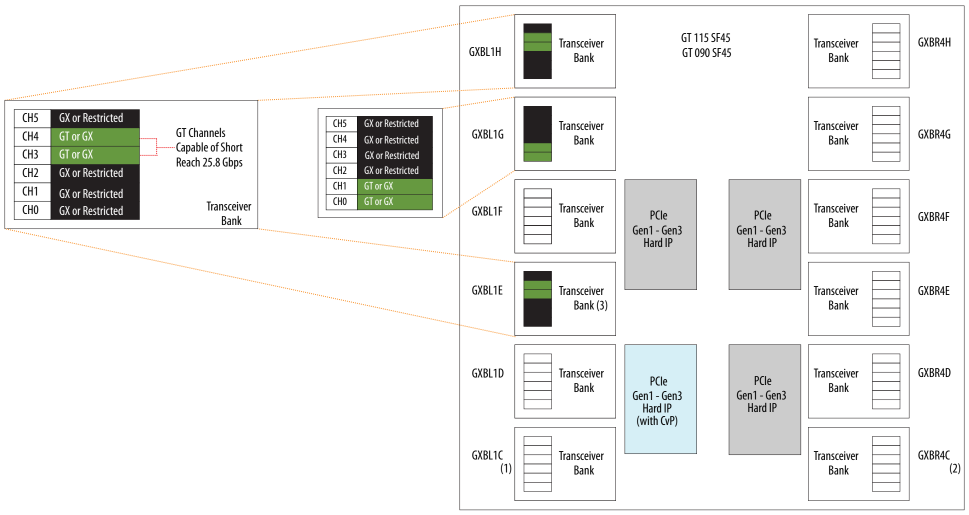
- Intel FPGA Arria 10 Transceiver PHY IP Core User Guide
  For information about the transceiver physical (PHY) layer architecture, PLLs, clock networks, and transceiver PHY IP.
- Intel Cyclone 10 GX Transceiver PHY User Guide
  For information about the transceiver PHY layer architecture, PLLs, clock networks, and transceiver PHY IP.

4.2. Hard IP Block Placement In Intel Arria 10 Devices

Intel Arria 10 devices include 1–4 hard IP blocks for PCI Express. The bottom left hard IP block includes the CvP functionality for flip chip packages. For other package types, the CvP functionality is in the bottom right block.

Note: Intel Arria 10 devices do not support configurations that configure a bottom (left or right) hard IP block with a Gen3 x4 or Gen3 x8 IP core and also configure the top hard IP block on the same side with a Gen3 x1 or Gen3 x2 IP core variation.

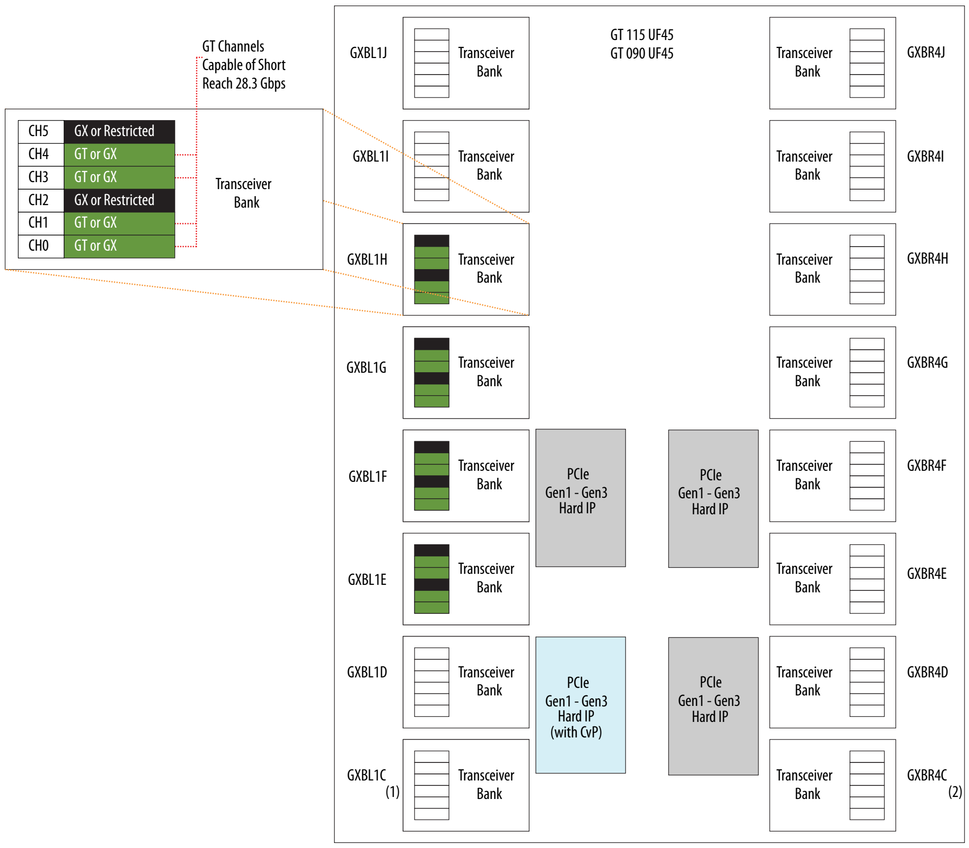
Figure 16. Intel Arria 10 Devices with 72 Transceiver Channels and Four PCIe Hard IP Blocks



Notes:
(1) Nomenclature of left column bottom transceiver banks always end with "C".
(2) Nomenclature of right column bottom transceiver banks may end with "C", "D", or "E".
(3) If a GT channel is used in transceiver bank GXBL1E, the PCIe Hard IP adjacent to GXBL1F and GXBL1E cannot be used.

Legend:
- 🟩 GT transceiver channels (channel 0, 1, 3, and 4).
- ⬛ GX transceiver channels (channel 2 and 5) with usage restrictions.
- ⬜ GX transceiver channels without usage restrictions.
- 🟦 PCIe Gen1 - Gen3 Hard IP blocks with Configuration via Protocol (CvP) capabilities.
- ▨ PCIe Gen1 - Gen3 Hard IP blocks without Configuration via Protocol (CvP) capabilities.

Figure 17. Intel Arria 10 Devices with 96 Transceiver Channels and Four PCIe Hard IP Blocks



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

Legend:

🟩 GT transceiver channels (channel 0, 1, 3, and 4)

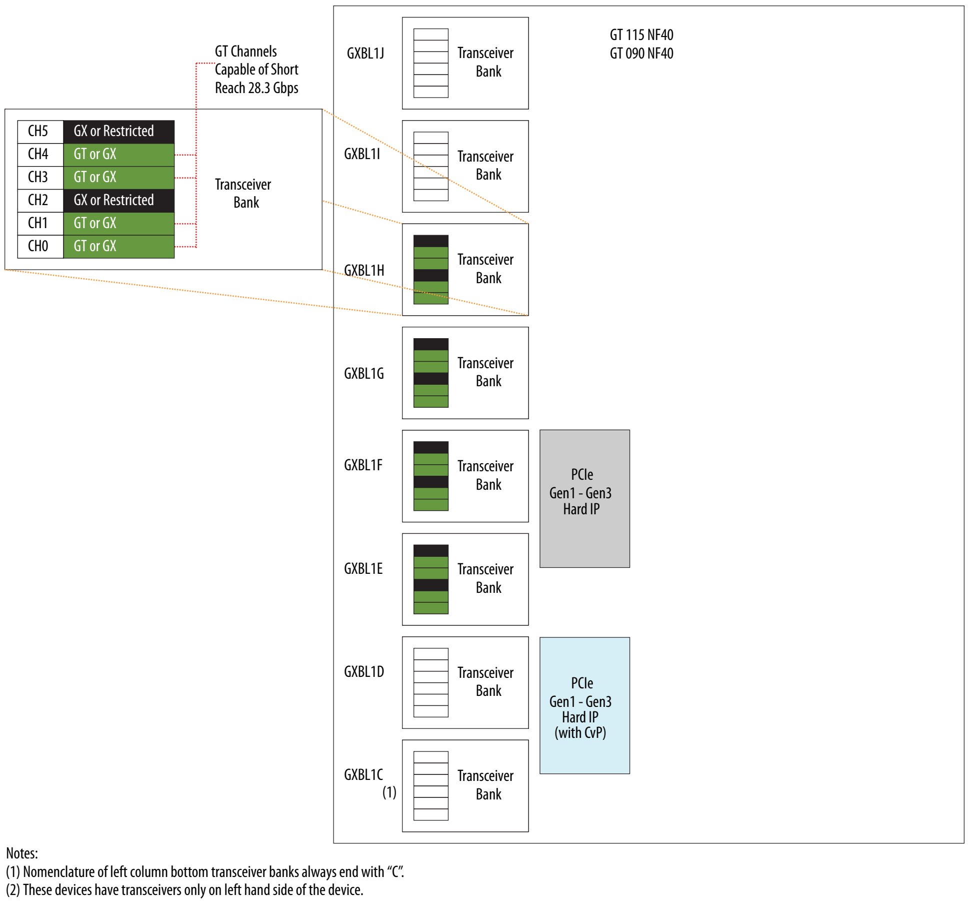
⬛ GX transceiver channels (channel 2 and 5) with usage restrictions.

⬜ GX transceiver channels without usage restrictions.

🟦 PCIe Gen1 - Gen3 Hard IP blocks with Configuration via Protocol (CvP) capabilities.

▨ PCIe Gen1 - Gen3 Hard IP blocks without Configuration via Protocol (CvP) capabilities.

Figure 18. Intel Arria 10 GT Devices with 48 Transceiver Channels and Two PCIe Hard IP Blocks


GT Channels Capable of Short Reach 28.3 Gbps
CH5 GX or Restricted
CH4 GT or GX
CH3 GT or GX
CH2 GX or Restricted
CH1 GT or GX
CH0 GT or GX
Transceiver Bank
GT 115 NF40
GT 090 NF40
GXBL1J
GXBL1I
GXBL1H
GXBL1G
GXBL1F
GXBL1E
GXBL1D
GXBL1C (1)
PCIe Gen1 - Gen3 Hard IP
PCIe Gen1 - Gen3 Hard IP (with CvP)
Notes:
(1) Nomenclature of left column bottom transceiver banks always end 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).
⬛ GX transceiver channels (channel 2 and 5) with usage restrictions.
⬜ GX transceiver channels without usage restrictions.
🟦 PCIe Gen3 HIP blocks with Configuration via Protocol (CvP) capabilities.
▨ PCIe Gen3 HIP blocks without Configuration via Protocol (CvP) capabilities.

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

Related Information

Intel FPGA Arria 10 Transceiver PHY IP Core User Guide
For information about the transceiver physical (PHY) layer architecture, PLLs, clock networks, and transceiver PHY IP.

4.3. Channel and Pin Placement for the Gen1, Gen2, and Gen3 Data Rates

The following figures illustrate pin placements for the Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express.

In these figures, channels that are not used for the PCI Express protocol are available for other protocols. Unused channels are shown in gray.

Note: In all configurations, physical channel 4 in the PCS connects to logical channel 0 in the hard IP. You cannot change the channel placements illustrated below.

For the possible values of *<txvr_block_N>* and *<txvr_block_N+1>*, refer to the figures that show the physical location of the Hard IP PCIe blocks in the different types of Intel Arria 10 or Intel Cyclone 10 GX devices, at the start of this chapter. For each hard IP block, the transceiver block that is adjacent and extends below the hard IP block, is *<txvr_block_N>*, and the transceiver block that is directly above is *<txvr_block_N+1>* . For example, in an Intel Arria 10 device with 96 transceiver channels and four PCIe hard IP blocks, if your design uses the hard IP block that supports CvP, *<txvr_block_N>* is GXB1C and *<txvr_block_N+1>* is GXB1D.

Note: Intel Cyclone 10 GX devices support x1, x2, and x4 at the Gen1 and Gen2 data rates.

Figure 19. Gen1, Gen2, and Gen3 x1 Channel and Pin Placement

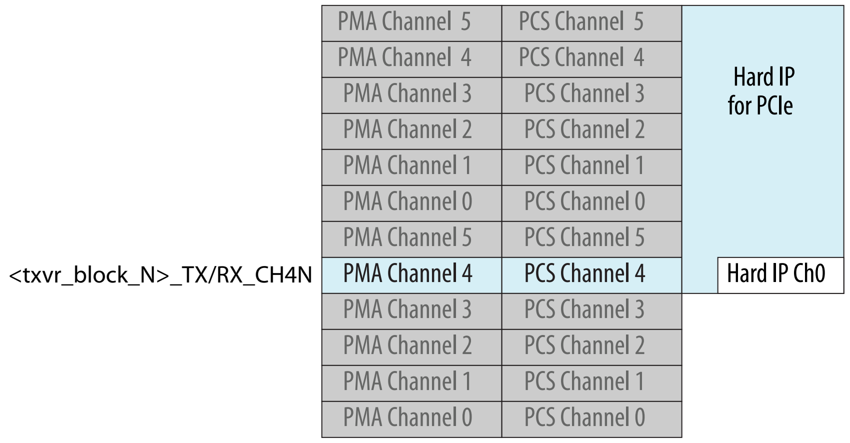


Figure 20. Gen1 Gen2, and Gen3 x2 Channel and Pin Placement

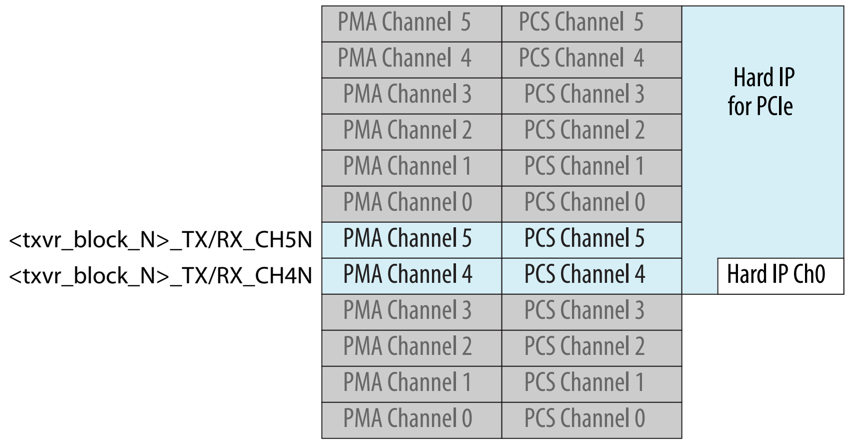


Figure 21. Gen1, Gen2, and Gen3 x4 Channel and Pin Placement

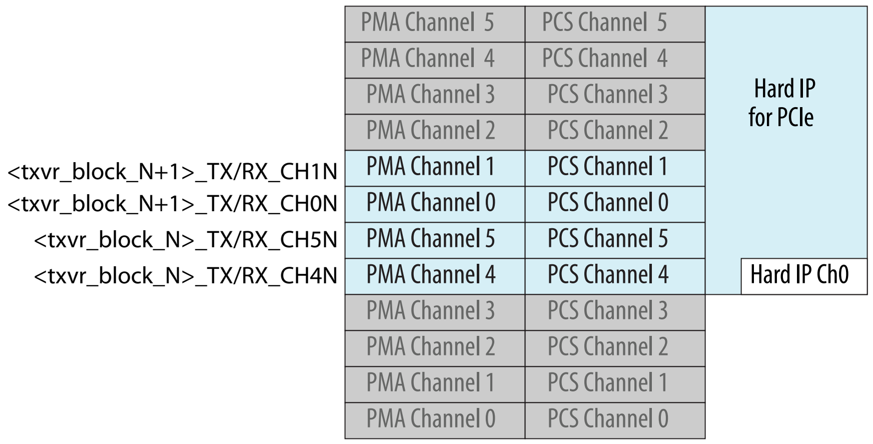
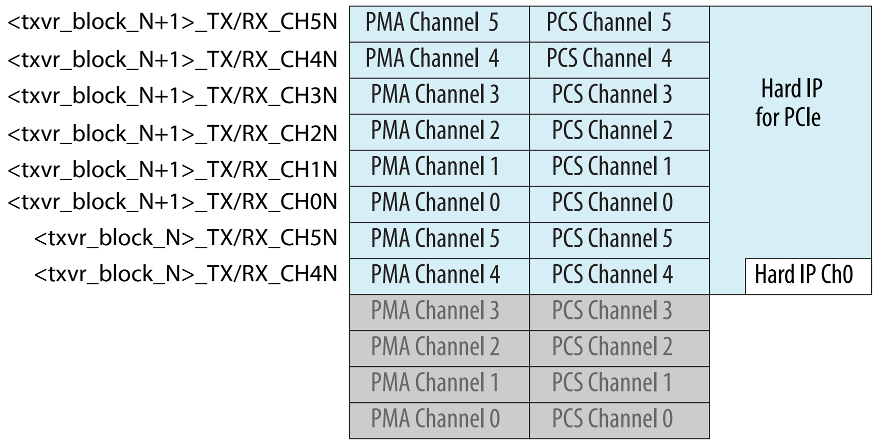


Figure 22. Gen1, Gen2, and Gen3 x8 Channel and Pin Placement



4.4. Channel Placement and fPLL and ATX PLL Usage for the Gen3 Data Rate

The following figures illustrate the channel placement for the Intel Arria 10 Hard IP for PCI Express.

Gen3 variants must initially train at the Gen1 data rate. Consequently, Gen3 variants require an fPLL to generate the 2.5 and 5.0 Gbps clocks, and an ATX PLL to generate the 8.0 Gbps clock. In these figures, channels that are not used for the PCI Express protocol are available for other protocols. Unused channels are shown in gray.

Note: In all configurations, physical channel 4 in the PCS connects to logical channel 0 in the hard IP. You cannot change the channel placements illustrated below.

Figure 23. Intel Arria 10 Gen3 x1 Channel Placement

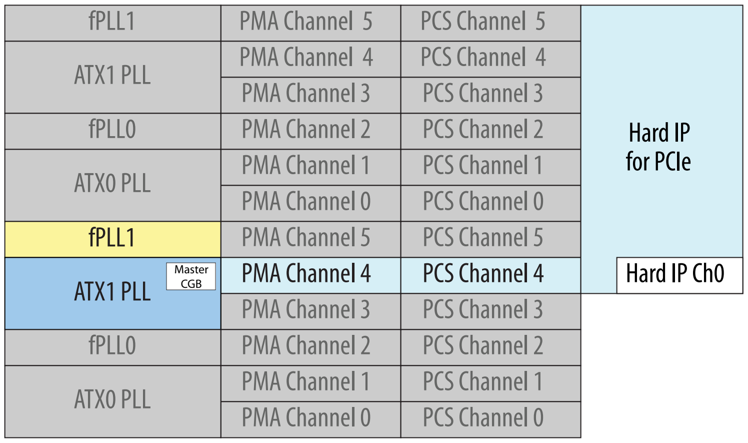


Figure 24. Intel Arria 10 Gen3 x2 Channel Placement

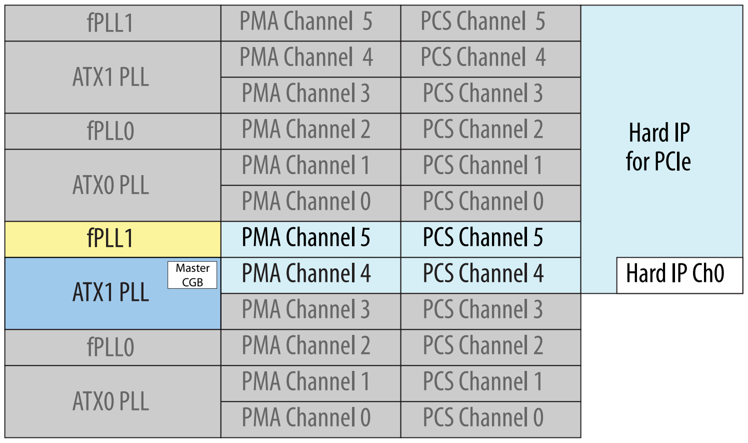


Figure 25. Intel Arria 10 Gen3 x4 Channel Placement

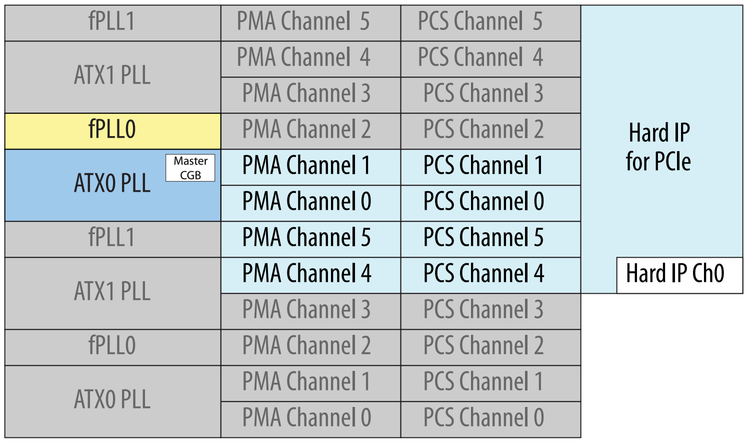
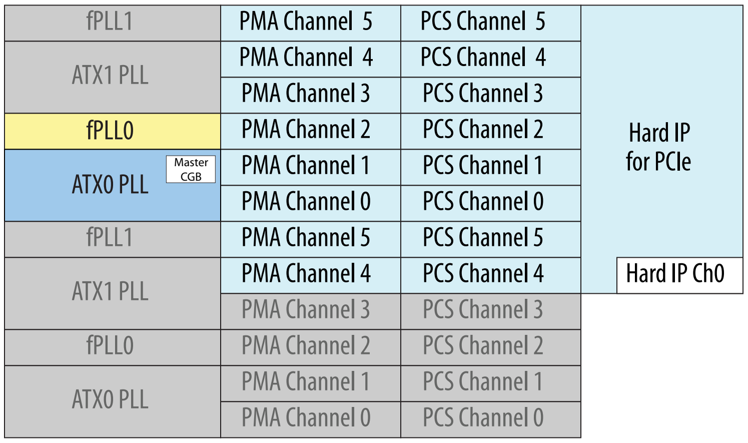


Figure 26. Gen3 x8 Channel Placement



4.5. PCI Express Gen3 Bank Usage Restrictions

Any transceiver channels that share a bank with active PCI Express interfaces that are Gen3 capable have the following restrictions. This includes both Hard IP and Soft IP implementations:

- When VCCR_GXB and VCCT_GXB are set to 1.03 V or 1.12 V, the maximum data rate supported for the non-PCIe channels in those banks is 12.5 Gbps for chip-to-chip applications. These channels cannot be used to drive backplanes or for GT rates.

PCI Express interfaces that are only Gen1 or Gen2 capable are not affected.

Status

Affects all Intel Arria 10 ES and production devices. No fix is planned.

5. 64- or 128-Bit Avalon-MM Interface to the Endpoint Application Layer

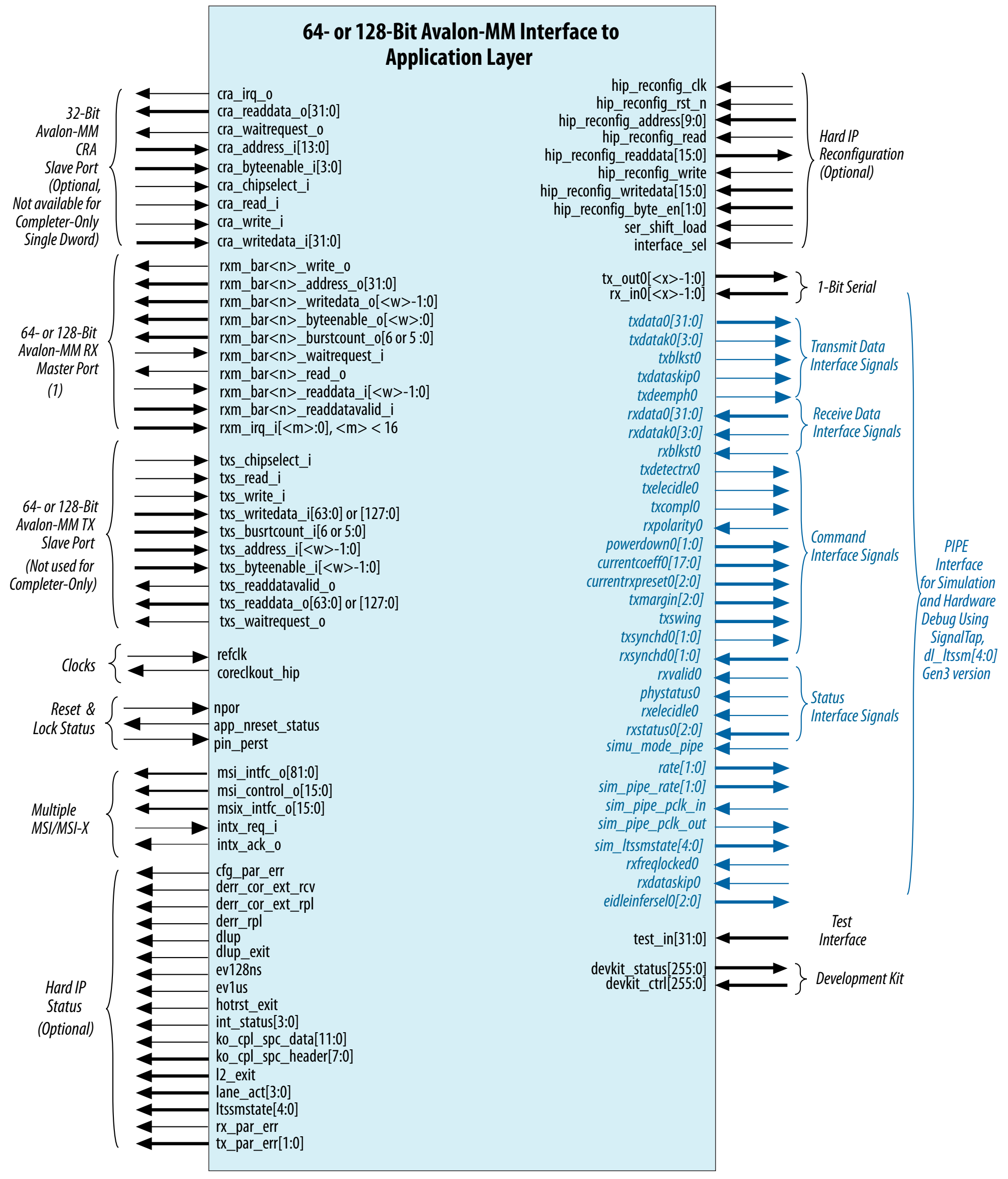
The Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express with an Avalon-MM interface to the Application Layer includes an Avalon-MM bridge. This bridge translates PCI Express TLPs to standard Avalon-MM read and write commands, and vice versa. Consequently, you do not need a detailed understanding of the PCI Express TLPs to use this Avalon-MM variant.

The Avalon-MM Intel Arria 10 Hard IP for PCI Express communicates with the Application Layer in the FPGA core fabric via the following interfaces:

- RX Master (RXM): This is a bursting RX Avalon-MM master interface that translates Memory Read and Write TLPs from the PCIe domain to Avalon-MM reads and writes and sends them to the slave in the Avalon-MM memory space.
- TX Slave (TXS): This is a bursting TX Avalon-MM slave interface that translates memory-mapped reads and writes from the Avalon-MM domain to PCIe Memory Read and Write TLPs and sends them to the PCIe memory space.
- Control Register Access (CRA): This optional Avalon-MM slave interface allows the Application Layer logic to access the internal control and status registers of the IP core.
- Hard IP Reconfiguration: This optional interface allows the Application Layer logic to dynamically modify the contents of the IP core's configuration registers that are read-only at run time.
- Hard IP Status: This optional interface contains status signals for the Hard IP to facilitate the debugging process.
- MSI/MSI-X: These interfaces provide the necessary information for the Application Layer logic to construct and send Message Signaled Interrupts to the host.

Note: The PIPE interface is used to communicate with the PHY Layer and not the Application Layer.

Figure 27. Signals in 64- or 128-Bit Avalon-MM Interface to the Application Layer



(1) n = 0, 1,2,3,4, or 5. Signals corresponding to the BARs enabled in the parameter editor will show up on the symbol.

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
For information about the Avalon-MM interface protocol.

5.1. 32-Bit Non-Bursting Avalon-MM Control Register Access (CRA) Slave Signals

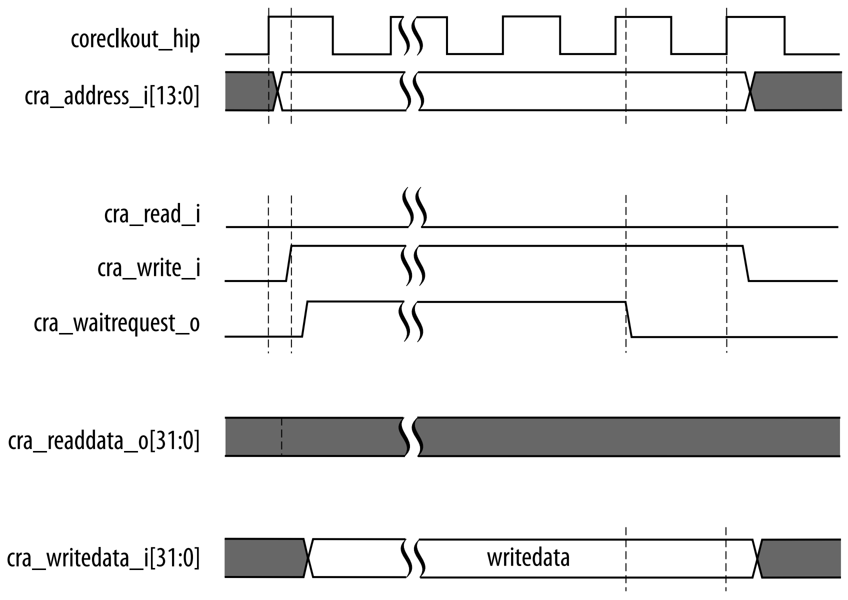
The optional CRA port for the full-featured IP core allows upstream PCI Express devices and external Avalon-MM masters to access internal control and status registers. Both Endpoint and Root Port applications can use the CRA interface.

Table 25. Avalon-MM CRA Slave Interface Signals

Signal Name	Direction	Description
`cra_irq_O`	Output	Interrupt request. A port request for an Avalon-MM interrupt.
`cra_readdata_o[31:0]`	Output	Read data lines.
`cra_waitrequest_o`	Output	Wait request to hold off more requests.
`cra_address_i[13:0]`	Input	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. To read or write individual bytes of a dword, use byte enables. For example, to write bytes 0 and 1, set `cra_byteenable_i[3:0]= 4'b0011` . Refer to *Valid Byte Enable Configurations* for valid byte enable patterns.
`cra_byteenable_i[3:0]`	Input	Byte enable.
`cra_chipselect_i`	Input	Chip select signal to this slave.
`cra_read_i`	Input	Read enable.
`cra_write_i`	Input	Write request.
`cra_writedata_i[31:0]`	Input	Write data.

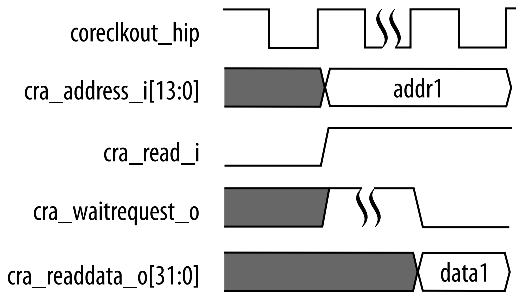
The CRA write request uses the high to low transition of `CraWaitRequest_o` to signal transaction completion

Figure 28. CRA Write Transaction



The CRA read transaction has similar timings to the CRA write transaction. The `CraReadData_o[31:0]` signals are valid at the clock cycle when `CraWaitRequest_o` is low. You can use the first rising clock edge after `CraWaitRequest_o` goes low to latch the data.

Figure 29. CRA Read Transaction



Related Information

PCI Express-to-Avalon-MM Downstream Write Requests on page 168

5.2. Bursting and Non-Bursting Avalon-MM Module Signals

The Avalon-MM Master module translates read and write TLPs received from the PCIe link to Avalon-MM transactions for connected slaves. You can enable up to six Avalon-MM Master interfaces. One of the six Base Address Registers (BARs) define the base address for each master interface. This module allows other PCIe components, including host software, to access the Avalon-MM slaves connected in the Platform Designer.

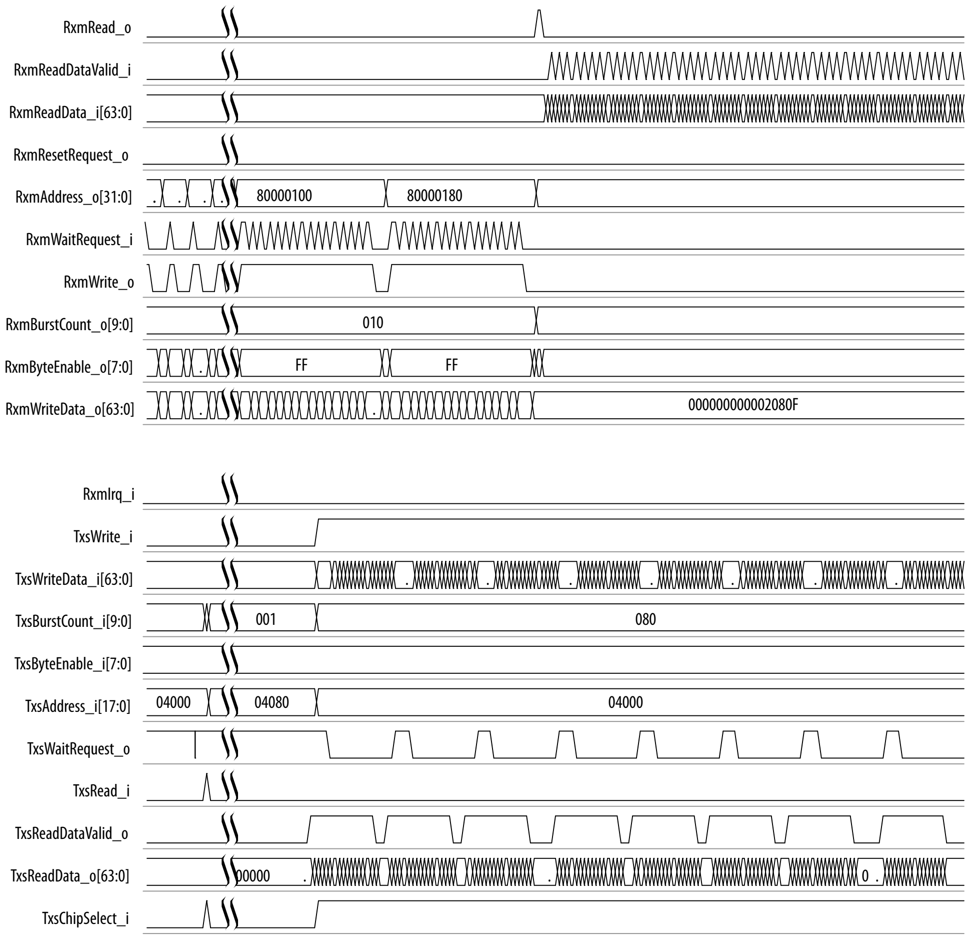
Table 26. Avalon-MM RX Master Interface Signals

<*n*> = the BAR number, and can be 0, 1, 2, 3, 4, or 5.

Signal Name	Direction	Description
`rxm_bar<n>_write_o`	Output	Asserted by the core to request a write to an Avalon-MM slave.
`rxm_bar<n>_address_o[<W>-1:0]`	Output	The address of the Avalon-MM slave being accessed.
`rxm_bar<n>_writedata_o[<w>-1:0]`	Output	RX data being written to slave. <*w*> = 64 or 128 for the full-featured IP core. <*w*> = 32 for the completer-only IP core.
`rxm_bar<n>_byteenable_o[<w>-1:0 ]`	Output	Dword enables for write data.
`rxm_bar<n>_burstcount_o[6 or 5:0]` (available in burst mode only)	Output	>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. This optional signal is available for BAR2 only when you turn on **Enable burst capabilities for RXM BAR2 ports.**
`rxm_bar<n>_waitrequest_i`	Input	Asserted by the external Avalon-MM slave to hold data transfer.
`rxm_bar<n>_read_o`	Output	Asserted by the core to request a read.
`rxm_bar<n>_readdata_i[<w>-1:0]`	Input	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 or 128 for the full-featured IP core. <*w*> = 32 for the completer-only IP core.
`rxm_bar<n>_readdatavalid_i`	Input	Asserted by the system interconnect fabric to indicate that the read data is valid.
`rxm_irq_i[<m>:0], <m> < 16`	Input	Connect interrupts to the Avalon-MM interface. These signals are only available for the Avalon-MM when the CRA port is enabled. A rising edge triggers an MSI interrupt. The hard IP core converts this event to an MSI interrupt and sends it to the Root Port. The host reads the `Interrupt Status` register to retrieve the interrupt vector. Host software services the interrupt and notifies the target upon completion. As many as 16 individual interrupt signals (<*m*>≤15) are available. 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:* These signals are not available when the IP core is operating in DMA mode (i.e. when the **Enable Avalon-MM DMA** option in the **Avalon-MM Settings** tab of the GUI is set to **On**).

The following timing diagram illustrates the RX master port propagating requests to the Application Layer and also shows simultaneous read and write activities.

Figure 30. Simultaneous RXM Read and RXM Write



5.3. 64- or 128-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 Intel Arria 10 or Intel Cyclone 10 GX 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, Intel recommends using a read request size of 128 bytes. A 512-byte read request results in 2, 256-byte TLPs with delays until all 256 bytes are available. Performance analyses show that a 128-byte read request size results in the lowest latency for typical systems.

Table 27. Avalon-MM TX Slave Interface Signals

Signal Name	Direction	Description
`txs_chipselect_i`	Input	The system interconnect fabric asserts this signal to select the TX slave port.
`txs_read_i`	Input	Read request asserted by the system interconnect fabric to request a read.

continued...

Signal Name	Direction	Description
`txs_write_i`	Input	Write request asserted by the system interconnect fabric to request a write.
`txs_writedata_i[127 or 63:0]`	Input	Write data sent by the external Avalon-MM master to the TX slave port.
`txs_burstcount_i[6 or 5:0]`	Input	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_i[<w>-1:0]`	Input	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_i[<w>-1:0]`	Input	Byte enables for read and write 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. For the 128-bit interface, the following restrictions apply: • All bytes of a single dword must either be enabled or disabled. • If more than 1 dword is enabled, the enabled dwords must be contiguous. The following patterns are legal: — 16'hF000 — 16'h0F00 — 16'h00F0 — 16'h000F — 16'hFF00 — 16'h0FF0 — 16'h00FF — 16'hFFF0 — 16'h0FFF — 16'hFFFF
`txs_readdatavalid_o`	Output	Asserted by the bridge to indicate that read data is valid.
`txs_readdata_o[127 or 63:0]`	Output	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`	Output	Asserted by the bridge to hold off read or write data when running out of buffer space. If this signal is asserted during an operation, the master should maintain the read or write signal and write data stable until after the wait request is deasserted. > `txs_read_i` must be deasserted when is > `txs_waitrequest_o` asserted.

5.4. Clock Signals

Table 28. Clock Signals

Signal	Direction	Description
`refclk`	Input	Reference clock for the IP core. It must have the frequency specified under the **System Settings** heading in the parameter editor. This is a dedicated free running input clock to the dedicated REFCLK pin.
`coreclkout_hip`	Output	This is a fixed frequency clock used by the Data Link and Transaction Layers.

Related Information

Clocks on page 95

5.5. Reset, Status, and Link Training Signals

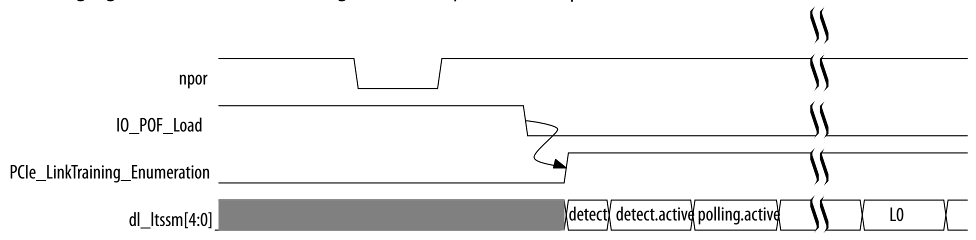
Refer to *Reset and Clocks* for more information about the reset sequence and a block diagram of the reset logic.

Table 29. Reset Signals

Signal	Direction	Description
npor	Input	Active low reset signal. In the Intel 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. Resets the entire IP Core and transceiver. Asynchronous. 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 reset controller, refer to *Reset*.
app_nreset_status	Output	Active low reset signal. It is derived from npor or pin_perstn. You can use this signal to reset the Application Layer.
pin_perst	Input	Active low reset from the PCIe reset pin of the device. pin_perst resets the datapath and control registers. Configuration via Protocol (CvP) requires this signal. For more information about CvP refer to *Arria 10 CvP Initialization and Partial Reconfiguration via Protocol User Guide*. Intel Arria 10 devices can have up to 4 instances of the Hard IP for PCI Express IP core. Each instance has its own pin_perst signal. Intel Cyclone 10 GX have a single instance of the Hard IP for PCI Express IP core. *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 Intel Arria 10 or Intel Cyclone 10 GX device, Intel recommends that you use the bottom left Hard IP first. This is the only location that supports CvP over a PCIe link. If your design does not require CvP, you may select other Hard IP blocks. Refer to the *Arria 10 GX, GT, and SX Device Family Pin Connection Guidelines* or *Intel Cyclone 10 GX Device Family Pin Connection Guidelines* for more detailed information about these pins.

Figure 31. 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 CvP and a 32-bit data width (FPP x32) or use the Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express in autonomous mode.

Related Information

- PCI Express Card Electromechanical Specification 2.0
- Arria 10 CvP Initialization and Partial Reconfiguration over PCI Express User Guide

5.6. Interrupts for Endpoints when Multiple MSI/MSI-X Support Is Enabled

Application Layer logic must construct the MSI (MemWr) TLP and send it using the TX slave (TXS) interface. For designs supporting multiple MSI/MSI-X, use the signals described below. For designs using a MSI TLP, use the control register access (CRA) interface to read the MSI Capability registers. The MSI information is at address offsets 14'h3C24, 14'h3C28, 14'h3C54, and 14'h3C5C. The Bus Master Enable bit is at address 14h'3C00.

Table 30. Exported Interrupt Signals for Endpoints when Multiple MSI/MSI-X Support is Enabled

The following table describes the IP core’s exported interrupt signals when you turn on **Enable multiple MSI/MSI-X support** under the **Avalon-MM System Settings** banner in the parameter editor.

Signal	Direction	Description
`msi_intfc_o[81:0]`	Output	This bus provides the following MSI address, data, and enabled signals: • `msi_intfc_o[81]`: Master enable • `msi_intfc_o[80}`: MSI enable • `msi_intfc_o[79:64]`: MSI data • `msi_intfc_o[63:0]`: MSI address
`msi_control_o[15:0]`	Output	Provides for system software control of MSI as defined in Section 6.8.1.3 *Message Control for MSI* in the *PCI Local Bus Specification, Rev. 3.0*. The following fields are defined: • `msi_control_o[15:9]`: Reserved • `msi_control_o[8]`: Per-vector masking capable • `msi_control_o[7]`: 64-bit address capable • `msi_control_o[6:4]`: Multiple Message Enable • `msi_control_o[3:1]`: MSI Message Capable • `msi_control_o[0]`: MSI Enable.

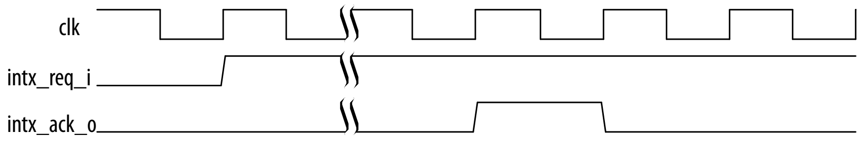
continued...

Signal	Direction	Description
`msix_intfc_o[15:0]`	Output	Provides for system software control of MSI-X as defined in Section 6.8.2.3 *Message Control for MSI-X* in the *PCI Local Bus Specification, Rev. 3.0*. The following fields are defined: • `msix_intfc_o[15]`: Enable • `msix_intfc_o[14]`: Mask • `msix_intfc_o[13:11]`: Reserved • `msix_intfc_o[10:0]`: Table size
`intx_req_i`	Input	When asserted, the Endpoint is requesting attention from the interrupt service routine unless MSI or MSI-X interrupts are enabled. Remains asserted until the device driver clears the pending request.
`intx_ack_o`	Output	This signal is the acknowledge for `IntxReq_i`. It is asserted for at least one cycle either when either of the following events occur: • The `Assert_INTA` message TLP has been transmitted in response to the assertion of the `IntxReq_i`. • The `Deassert_INTA` message TLP has been transmitted in response to the deassertion of the `IntxReq_i` signal. Refer to the timing diagrams below.

The following figure illustrates interrupt timing for the legacy interface. In this figure the assertion of `IntxReq_i` instructs the Hard IP for PCI Express to send an `Assert_INTA` message TLP.

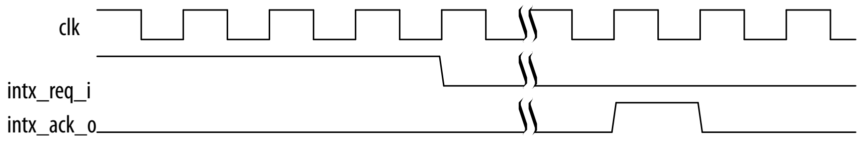
The following figure illustrates the timing for deassertion of legacy interrupts. The assertion of `IntxReq_i` instructs the Hard IP for PCI Express to send a `Deassert_INTA` message.

Figure 32. Legacy Interrupt Assertion



The following figure illustrates the timing for deassertion of legacy interrupts. The assertion of `IntxReq_i` instructs the Hard IP for PCI Express to send a `Deassert_INTA` message.

Figure 33. Legacy Interrupt Deassertion



5.7. Hard IP Status Signals

Table 31. Hard IP Status Signals

The following table describes additional status signals related to the reset function for the including the `ltsssm_state[4:0]` bus that indicates the current link training state. Use them to debug link training issues.

Signal	Direction	Description
`cfg_par_err`	Output	Indicates that a parity error in a TLP routed to the internal Configuration Space. This error is also logged in the Vendor Specific Extended Capability internal error register. You must reset the Hard IP if this error occurs.
`derr_cor_ext_rcv`	Output	Indicates a corrected error in the RX buffer. This signal is for debug only. It is not valid until the RX buffer is filled with data. This is a pulse, not a level, signal. Internally, the pulse is generated with the 500 MHz clock. A pulse extender extends the signal so that the FPGA fabric running at 250 MHz can capture it. Because the error was corrected by the IP core, no Application Layer intervention is required.
`derr_cor_ext_rpl`	Output	Indicates a corrected ECC error in the retry buffer. This signal is for debug only. Because the error was corrected by the IP core, no Application Layer intervention is required. (4)
`derr_rpl`	Output	Indicates an uncorrectable error in the retry buffer. This signal is for debug only. (4)
`dlup`	Output	When asserted, indicates that the Hard IP block is in the Data Link Control and Management State Machine (DLCMSM) DL_Up state.
`dlup_exit`	Output	This signal is asserted low for one `pld_clk` cycle when the IP core exits the DLCMSM DL_Up state, indicating that the Data Link Layer has lost communication with the other end of the PCIe link and left the Up state. When this pulse is asserted, the Application Layer should generate an internal reset signal that is asserted for at least 32 cycles.
`ev128ns`	Output	Asserted every 128 ns to create a time base aligned activity.
`ev1us`	Output	Asserted every 1 µs to create a time base aligned activity.
`hotrst_exit`	Output	Hot reset exit. This signal is asserted for 1 clock cycle when the LTSSM exits the hot reset state. This signal should cause the Application Layer to be reset. This signal is active low. When this pulse is asserted, the Application Layer should generate an internal reset signal that is asserted for at least 32 cycles.
`int_status[3:0]`	Output	These signals drive legacy interrupts to the Application Layer as follows: • int_status[0]: interrupt signal A • int_status[1]: interrupt signal B • int_status[2]: interrupt signal C • int_status[3]: interrupt signal D
`ko_cpl_spc_data`	Output	The Application Layer can use this signal to build circuitry to prevent RX buffer overflow for completion data. Endpoints must advertise infinite space for completion data; however, RX buffer space is finite. `ko_cpl_spc_data` is a static signal that reflects the total number of 16 byte completion data units that can be stored in the completion RX buffer.
`ko_cpl_spc_header`	Output	The Application Layer can use this signal to build circuitry to prevent RX buffer overflow for completion headers. Endpoints must advertise infinite space for completion headers; however, RX buffer space is finite. `ko_cpl_spc_header` is a static signal that indicates the total number of completion headers that can be stored in the RX buffer.

continued...

(5) Debug signals are not rigorously verified and should only be used to observe behavior. Debug signals should not be used to drive logic custom logic.

Signal	Direction	Description
`l2_exit`	Output	L2 exit. This signal is active low and otherwise remains high. It is asserted for one cycle (changing value from 1 to 0 and back to 1) after the LTSSM transitions from l2.idle to detect. When this pulse is asserted, the Application Layer should generate an internal reset signal that is asserted for at least 32 cycles.
`lane_act[3:0]`	Output	Lane Active Mode: This signal indicates the number of lanes that configured during link training. The following encodings are defined: • 4’b0001: 1 lane • 4’b0010: 2 lanes • 4’b0100: 4 lanes • 4’b1000: 8 lanes
`ltssmstate[4:0]`	Output	LTSSM state: The LTSSM state machine encoding defines the following states: • 00000: Detect.Quiet • 00001: Detect.Active • 00010: Polling.Active • 00011: Polling.Compliance • 00100: Polling.Configuration • 00101: Polling.Speed • 00110: config.Linkwidthstart • 00111: Config.Linkaccept • 01000: Config.Lanenumaccept • 01001: Config.Lanenumwait • 01010: Config.Complete • 01011: Config.Idle • 01100: Recovery.Rcvlock • 01101: Recovery.Rcvconfig • 01110: Recovery.Idle • 01111: L0 • 10000: Disable • 10001: Loopback.Entry • 10010: Loopback.Active • 10011: Loopback.Exit • 10100: Hot.Reset • 10101: L0s • 11001: L2.transmit.Wake • 11010: Speed.Recovery • 11011: Recovery.Equalization, Phase 0 • 11100: Recovery.Equalization, Phase 1 • 11101: Recovery.Equalization, Phase 2 • 11110: recovery.Equalization, Phase 3
`rx_par_err`	Output	When asserted for a single cycle, indicates that a parity error was detected in a TLP at the input of the RX buffer. This error is logged as an uncorrectable internal error in the VSEC registers. For more information,

continued...

Signal	Direction	Description
		refer to *Uncorrectable Internal Error Status Register*. If this error occurs, you must reset the Hard IP if this error occurs because parity errors can leave the Hard IP in an unknown state.
`tx_par_err[1:0]`	Output	When asserted for a single cycle, indicates a parity error during TX TLP transmission. These errors are logged in the VSEC register. The following encodings are defined: • 2'b10: A parity error was detected by the TX Transaction Layer. The TLP is nullified and logged as an uncorrectable internal error in the VSEC registers. For more information, refer to *Uncorrectable Internal Error Status Register*. • 2'b01: Some time later, the parity error is detected by the TX Data Link Layer which drives 2'b01 to indicate the error. Intel recommends resetting the Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express when this error is detected. Contact Intel if resetting becomes unworkable. Note that not all simulation models assert the Transaction Layer error bit in conjunction with the Data Link Layer error bit.

Related Information

- PCI Express Card Electromechanical Specification 2.0
- Documentation: Device Pin Connection Guidelines

5.7.1. Hard IP Reconfiguration Interface

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

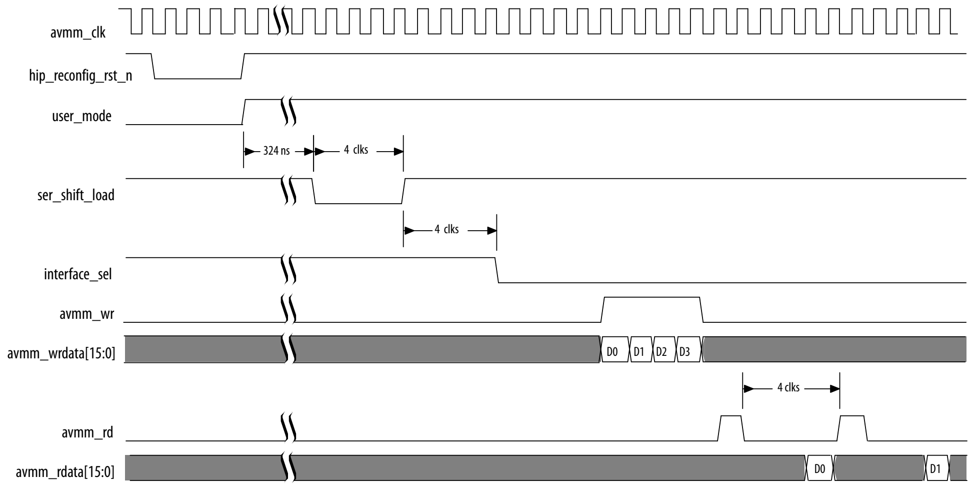
Table 32. Hard IP Reconfiguration Signals

Signal	Direction	Description
`hip_reconfig_clk`	Input	Reconfiguration clock. The frequency range for this clock is 100–125 MHz.
`hip_reconfig_rst_n`	Input	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]`	Input	The 10-bit reconfiguration address.
`hip_reconfig_read`	Input	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]`	Output	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`	Input	Write signal.
`hip_reconfig_writedata[15:0]`	Input	16-bit write model.

continued...

Signal	Direction	Description
hip_reconfig_byte_en[1:0]	Input	Byte enables, currently unused.
ser_shift_load	Input	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	Input	A selector which must be asserted when performing dynamic reconfiguration. Drive this signal low 4 clock cycles after the release of ser_shif t_load.

Figure 34. Hard IP Reconfiguration Bus Timing of Read-Only Registers



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

Related Information

Avalon Interface Specifications
For information about the Avalon-MM interfaces to implement read and write interfaces for master and slave components.

5.8. Physical Layer Interface Signals

Intel provides an integrated solution with the Transaction, Data Link and Physical Layers. The IP Parameter Editor generates a SERDES variation file, <*variation*>_serdes.v or **.vhd** , in addition to the Hard IP variation file, <*variation*>.v or .vhd. The SERDES entity is included in the library files for PCI Express.

5.8.1. Serial Data Signals

This differential, serial interface is the physical link between a Root Port and an Endpoint.

The Intel Cyclone 10 GX PCIe IP Core supports 1, 2, or 4 lanes. Each lane includes a TX and RX differential pair. Data is striped across all available lanes.

Send Feedback

The Intel Arria 10 PCIe IP Core supports 1, 2, 4 or 8 lanes. Each lane includes a TX and RX differential pair. Data is striped across all available lanes.

Table 33. 1-Bit Interface Signals

In the following table <*n*> is the number of lanes.

Signal	Direction	Description
`tx_out[<n>-1:0]`	Output	Transmit output. These signals are the serial outputs of lanes <*n*>-*1*–0.
`rx_in[<n>-1:0]`	Input	Receive input. These signals are the serial inputs of lanes <*n*>-*1*–0.

Refer to *Pin-out Files for Intel Devices* for pin-out tables for all Intel 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 Intel Devices*.

Related Information

- Hard IP Block Placement In Intel Arria 10 Devices on page 37
- Hard IP Block Placement In Intel Cyclone 10 GX Devices on page 36
- Pin-out Files for Intel Devices

5.8.2. PIPE Interface Signals

These PIPE signals are available for Gen1, Gen2, and Gen3 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 . By default, the PIPE interface data width is 8 bits for Gen1 and Gen2 and 32 bits for Gen3. 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 Signal Tap.

Intel Cyclone 10 GX devices do not support the Gen3 data rate.

Note: The Intel Root Port BFM bypasses Gen3 Phase 2 and Phase 3 Equalization. However, Gen3 variants can perform Phase 2 and Phase 3 equalization if instructed by a third-party BFM.

In the following table, signals that include lane number 0 also exist for lanes 1-4. For Gen1 and Gen2 operation outputs can be left floating.

Table 34. PIPE Interface Signals

Signal	Direction	Description
`txdata0[31:0]`	Output	Transmit data <*n*>. This bus transmits data on lane <*n*>.
`txdatak0[3:0]`	Output	Transmit data control <*n*>. This signal serves as the control bit for `txdata` <*n*>. Bit 0 corresponds to the lowest-order byte of `txdata`, and so on. A value of 0 indicates a data byte. A value of 1 indicates a control byte. For Gen1 and Gen2 only.
`txblkst0`	Output	For Gen3 operation, indicates the start of a block in the transmit direction.
`txcompl0`	Output	Transmit compliance <*n*>. This signal forces the running disparity to negative in Compliance Mode (negative COM character).
`txdataskip0`	Output	For Gen3 operation. Allows the MAC to instruct the TX interface to ignore the TX data interface for one clock cycle. The following encodings are defined: • 1’b0: TX data is invalid • 1’b1: TX data is valid
`txdeemph0`	Output	Transmit de-emphasis selection. The Intel 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.
`txdetectrx0`	Output	Transmit detect receive <*n*>. This signal tells the PHY layer to start a receive detection operation or to begin loopback.
`txelecidle0`	Output	Transmit electrical idle <*n*>. This signal forces the TX output to electrical idle.
`txswing`	Output	When asserted, indicates full swing for the transmitter voltage. When deasserted indicates half swing.
`txmargin[2:0]`	Output	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.
`txsynchd0[1:0]`	Output	For Gen3 operation, specifies the transmit block type. The following encodings are defined: • 2'b01: Ordered Set Block • 2'b10: Data Block Designs that do not support Gen3 can let this signal float.
`rxdata0[31:0]`	Input	Receive data <*n*>. This bus receives data on lane <*n*>.
`rxdatak[3:0]`	Input	Receive data >*n*>. This bus receives data on lane <*n*>. Bit 0 corresponds to the lowest-order byte of `rxdata`, and so on. A value of 0 indicates a data byte. A value of 1 indicates a control byte. For Gen1 and Gen2 only.
`rxblkst0`	Input	For Gen3 operation, indicates the start of a block in the receive direction.
`rxdataskip0`	Output	For Gen3 operation. Allows the PCS to instruct the RX interface to ignore the RX data interface for one clock cycle. The following encodings are defined: • 1’b0: RX data is invalid • 1’b1: RX data is valid
`rxelecidle0`	Input	Receive electrical idle <*n*>. When asserted, indicates detection of an electrical idle.
`rxpolarity0`	Output	Receive polarity <*n*>. This signal instructs the PHY layer to invert the polarity of the 8B/10B receiver decoding block.
`rxstatus0[2:0]`	Input	Receive status <*n*>. This signal encodes receive status, including error codes for the receive data stream and receiver detection.

continued...

Signal	Direction	Description
`rxsynchd0[1:0]`	Input	For Gen3 operation, specifies the receive block type. The following encodings are defined: • 2'b01: Ordered Set Block • 2'b10: Data Block Designs that do not support Gen3 can ground this signal.
`rxvalid0`	Input	Receive valid <*n*>. This signal indicates symbol lock and valid data on `rxdata<n>` and `rxdatak <n>`.
`phystatus0`	Input	PHY status <*n*>. This signal communicates completion of several PHY requests.
`powerdown0[1:0]`	Output	Power down <*n*>. This signal requests the PHY to change its power state to the specified state (P0, P0s, P1, or P2).
`currentcoeff0[17:0]`	Output	For Gen3, specifies the coefficients to be used by the transmitter. The 18 bits specify the following coefficients: • [5:0]: C_{-1} • [11:6]: C_0 • [17:12]: C_{+1}
`currentrxpreset0[2:0]`	Output	For Gen3 designs, specifies the current preset.
`simu_mode_pipe`	Input	When set to 1, the PIPE interface is in simulation mode.
`sim_pipe_rate[1:0]`	Output	The 2-bit encodings have the following meanings: • 2’b00: Gen1 rate (2.5 Gbps) • 2’b01: Gen2 rate (5.0 Gbps) • 2’b10: Gen3 rate (8.0 Gbps)
`rate[1:0]`	Output	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`	Input	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_ltssmstate0[4:0]`	Input and Output	LTSSM state: The LTSSM state machine encoding defines the following states: • 5’b00000: Detect.Quiet • 5’b00001: Detect.Active • 5’b00010: Polling.Active • 5’b 00011: Polling.Compliance • 5’b 00100: Polling.Configuration • 5’b00101: Polling.Speed • 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

continued...

Signal	Direction	Description
		• 5’b10100: Hot.Reset • 5’b10101: L0s • 5’b11001: L2.transmit.Wake • 5’b11010: Recovery.Speed • 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`	Input	When asserted indicates that the `pclk_in` used for PIPE simulation is valid.
`eidleinfersel0[2:0]`	Output	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 in the 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

5.8.3. Intel Arria 10 Development Kit Conduit Interface

The Intel Arria 10 Development Kit conduit interface signals are optional signals that allow you to connect your design to the Intel Arria 10 FPGA Development Kit. Enable this interface by selecting **Enable Intel Arria 10 FPGA Development Kit connection** on the **Configuration, Debug, and Extension Options** tab of the component GUI. The `devkit_status` output port includes signals useful for debugging.

Table 35. The Intel Arria 10 Development Kit Conduit Interface

Signal Name	Direction	Description
`devkit_status[255:0]`	Output	The `devkit_status[255:0]` bus comprises the following status signals : • `devkit_status[1:0]`: `current_speed` • `devkit_status[2]`: `derr_cor_ext_rcv` • `devkit_status[3]`: `derr_cor_ext_rpl` • `devkit_status[4]`: `derr_err` • `devkit_status[5]`: `rx_par_err` • `devkit_status[7:6]`: `tx_par_err` • `devkit_status[8]`: `cfg_par_err` • `devkit_status[9]`: `dlup` • `devkit_status[10]`: `dlup_exit` • `devkit_status[11]`: `ev128ns` • `devkit_status[12]`: `ev1us` • `devkit_status[13]`: `hotrst_exit` • `devkit_status[17:14]`: `int_status[3:0]` • `devkit_status[18]`: `l2_exit` • `devkit_status[22:19]`: `lane_act[3:0]` • `devkit_status[27:23]`: `ltssmstate[4:0]` • `devkit_status[35:28]`: `ko_cpl_spc_header[7:0]`

continued...

Signal Name	Direction	Description
		• devkit_status[47:36]: ko_cpl_spc_data[11:0] • devkit_status[48]: rxfc_cplbuf_ovf • devkit_status[49]: reset_status • devkit_status[255:50]: Reserved
devkit_ctrl[255:0]	Input	The devkit_ctrl[255:0] bus comprises the following status signals. You can optionally connect these pins to an on-board switch for PCI-SIG compliance testing, such as bypass compliance testing. • devkit_ctrl[0]:test_in[0] is typically set to 1'b0 • devkit_ctrl[4:1]:test_in[4:1] is typically set to 4'b0100 • devkit_ctrl[6:5]:test_in[6:5] is typically set to 2'b01 • devkit_ctrl[31:7]:test_in[31:7] is typically set to 25'h3 • devkit_ctrl[63:32]:is typically set to 32'b0 • devkit_ctrl[255:64]:is typically set to 192'b0

5.8.4. Test Signals

Table 36. Test Interface Signals

The test_in bus provides run-time control and monitoring of the internal state of the IP core.

Signal	Direction	Description
test_in[31:0]	Input	The bits of the test_in bus have the following definitions. Set this bus to 0x00000188. • [0]: Simulation mode. This signal can be set to 1 to accelerate initialization by reducing the value of many initialization counters. • [1]: Reserved. Must be set to 1'b0. • [2]: Descramble mode disable. This signal must be set to 1 during initialization in order to disable data scrambling. You can use this bit in simulation for Gen1 and Gen2 Endpoints and Root Ports to observe descrambled data on the link. Descrambled data cannot be used in open systems because the link partner typically scrambles the data. • [4:3]: Reserved. Must be set to 2'b01. • [5]: Compliance test mode. Set this bit to 1'b0. Setting this bit to 1'b1 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. • [6]: Forces entry to compliance mode when a timeout is reached in the polling.active state and not all lanes have detected their exit condition. • [7]: Disable low power state negotiation. Intel recommends setting this bit. • [8]: Set this bit to 1'b1. • [31:9]: Reserved. Set to all 0s.

6. Registers

6.1. Correspondence between Configuration Space Registers and the PCIe Specification

Table 37. Address Map of Hard IP Configuration Space Registers

For the Type 0 and Type 1 Configuration Space Headers, the first line of each entry lists Type 0 values and the second line lists Type 1 values when the values differ.

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	N/A
0x050:0x05C	MSI Capability Structure	MSI Capability Structure
0x068:0x070	MSI-X Capability Structure	MSI-X Capability Structure
0x070:0x074	Reserved	N/A
0x078:0x07C	Power Management Capability Structure	PCI Power Management Capability Structure
0x080:0x0BC	PCI Express Capability Structure	PCI Express Capability Structure
0x0C0:0x0FC	Reserved	N/A
0x100:0x16C	Virtual Channel Capability Structure	Virtual Channel Capability
0x170:0x17C	Reserved	N/A
0x180:0x1FC	Virtual channel arbitration table	VC Arbitration Table
0x200:0x23C	Port VC0 arbitration table	Port Arbitration Table
0x240:0x27C	Port VC1 arbitration table	Port Arbitration Table
0x280:0x2BC	Port VC2 arbitration table	Port Arbitration Table
0x2C0:0x2FC	Port VC3 arbitration table	Port Arbitration Table
0x300:0x33C	Port VC4 arbitration table	Port Arbitration Table
0x340:0x37C	Port VC5 arbitration table	Port Arbitration Table
0x380:0x3BC	Port VC6 arbitration table	Port Arbitration Table
0x3C0:0x3FC	Port VC7 arbitration table	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	N/A
Overview of Configuration Space Register Fields		

continued...

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x000	Device ID, Vendor ID	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x004	Status, Command	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x008	Class Code, Revision ID	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x00C	BIST, Header Type, Primary Latency Timer, Cache Line Size	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x010	Base Address 0	Base Address Registers
0x014	Base Address 1	Base Address Registers
0x018	Base Address 2 Secondary Latency Timer, Subordinate Bus Number, Secondary Bus Number, Primary Bus Number	Base Address Registers Secondary Latency Timer, Type 1 Configuration Space Header, Primary Bus Number
0x01C	Base Address 3 Secondary Status, I/O Limit, I/O Base	Base Address Registers Secondary Status Register ,Type 1 Configuration Space Header
0x020	Base Address 4 Memory Limit, Memory Base	Base Address Registers Type 1 Configuration Space Header
0x024	Base Address 5 Prefetchable Memory Limit, Prefetchable Memory Base	Base Address Registers Prefetchable Memory Limit, Prefetchable Memory Base
0x028	Reserved Prefetchable Base Upper 32 Bits	N/A Type 1 Configuration Space Header
0x02C	Subsystem ID, Subsystem Vendor ID Prefetchable Limit Upper 32 Bits	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x030	I/O Limit Upper 16 Bits, I/O Base Upper 16 Bits	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x034	Reserved, Capabilities PTR	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x038	Reserved	N/A
0x03C	Interrupt Pin, Interrupt Line Bridge Control, Interrupt Pin, Interrupt Line	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x050	MSI-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	MSI-X 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
0x078	Capabilities Register Next Cap PTR Cap ID	PCI Power Management Capability Structure

continued...

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x07C	Data PM Control/Status Bridge Extensions Power Management Status & Control	PCI Power Management Capability Structure
0x080	PCI Express Capabilities Register Next Cap Ptr PCI Express Cap ID	PCI Express Capability Structure
0x084	Device Capabilities Register	PCI Express Capability Structure
0x088	Device Status Register Device Control Register	PCI Express Capability Structure
0x08C	Link Capabilities Register	PCI Express Capability Structure
0x090	Link Status Register Link Control Register	PCI Express Capability Structure
0x094	Slot Capabilities Register	PCI Express Capability Structure
0x098	Slot Status Register Slot Control Register	PCI Express Capability Structure
0x09C	Root Capabilities Register Root Control Register	PCI Express Capability Structure
0x0A0	Root Status Register	PCI Express Capability Structure
0x0A4	Device Capabilities 2 Register	PCI Express Capability Structure
0x0A8	Device Status 2 Register Device Control 2 Register	PCI Express Capability Structure
0x0AC	Link Capabilities 2 Register	PCI Express Capability Structure
0x0B0	Link Status 2 Register Link Control 2 Register	PCI Express Capability Structure
0x0B4:0x0BC	Reserved	PCI Express Capability Structure
0x800	Advanced Error Reporting 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

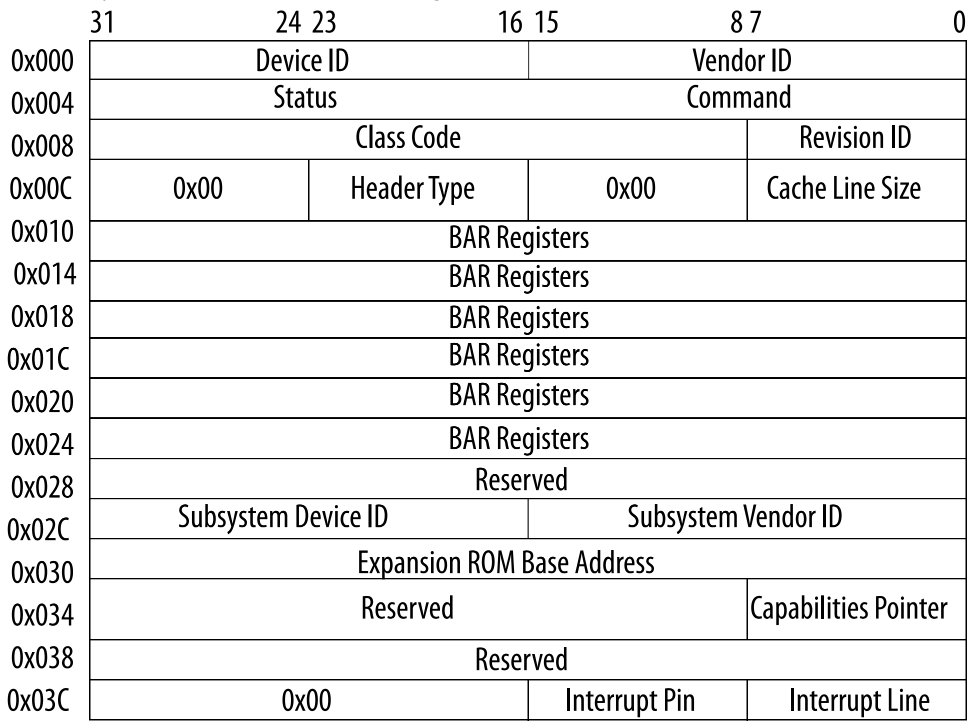
Related Information

PCI Express Base Specification 3.0

6.2. Type 0 Configuration Space Registers

Figure 35. Type 0 Configuration Space Registers - Byte Address Offsets and Layout

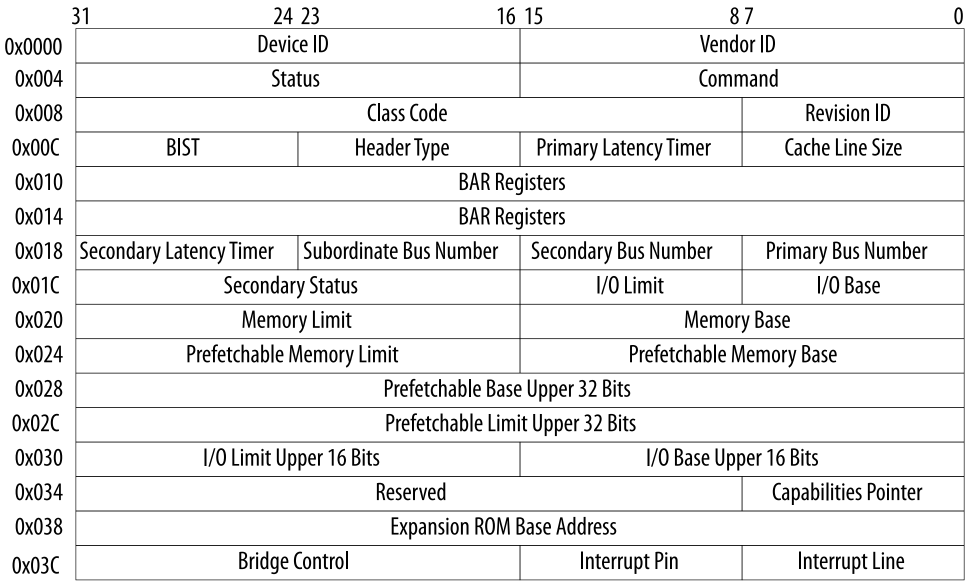
Endpoints store configuration data in the Type 0 Configuration Space. The Correspondence between Configuration Space Registers and the PCIe Specification on page 64 lists the appropriate section of the *PCI Express Base Specification* that describes these registers.



	31 – 24	23 – 16	15 – 8	7 – 0
0x000	Device ID		Vendor ID	
0x004	Status		Command	
0x008	Class Code			Revision ID
0x00C	0x00	Header Type	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

6.3. Type 1 Configuration Space Registers

Figure 36. Type 1 Configuration Space Registers (Root Ports)



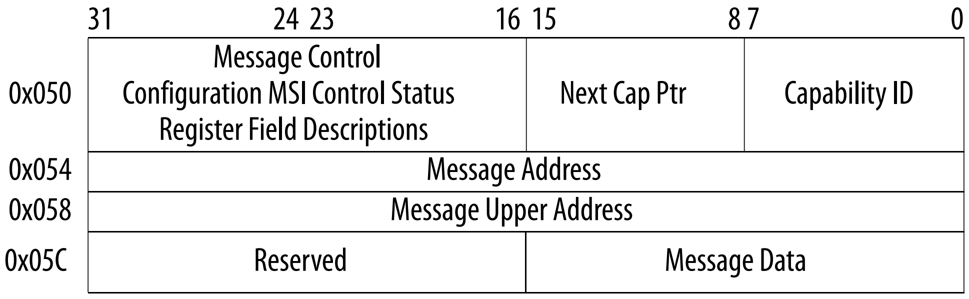
	31–24	23–16	15–8	7–0
0x0000	Device ID		Vendor ID	
0x004	Status		Command	
0x008	Class Code			Revision ID
0x00C	BIST	Header Type	Primary Latency Timer	Cache Line Size
0x010	BAR Registers			
0x014	BAR Registers			
0x018	Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number
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

Note: Avalon-MM DMA for PCIe does not support Type 1 configuration space registers.

6.4. PCI Express Capability Structures

The layout of the most basic Capability Structures are provided below. Refer to the *PCI Express Base Specification* for more information about these registers.

Figure 37. MSI Capability Structure



	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	

Figure 38. MSI-X Capability Structure

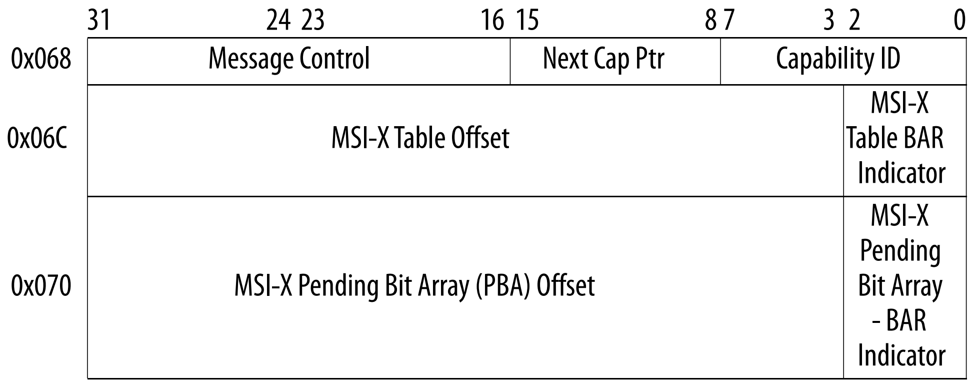

31
24 23
16 15
8 7
3 2
0
0x068
Message Control
Next Cap Ptr
Capability ID
0x06C
MSI-X Table Offset
MSI-X Table BAR Indicator
0x070
MSI-X Pending Bit Array (PBA) Offset
MSI-X Pending Bit Array - BAR Indicator

Figure 39. Power Management Capability Structure - Byte Address Offsets and Layout

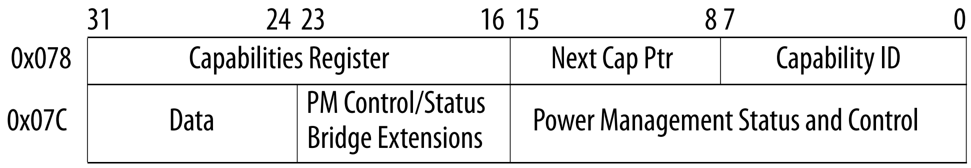


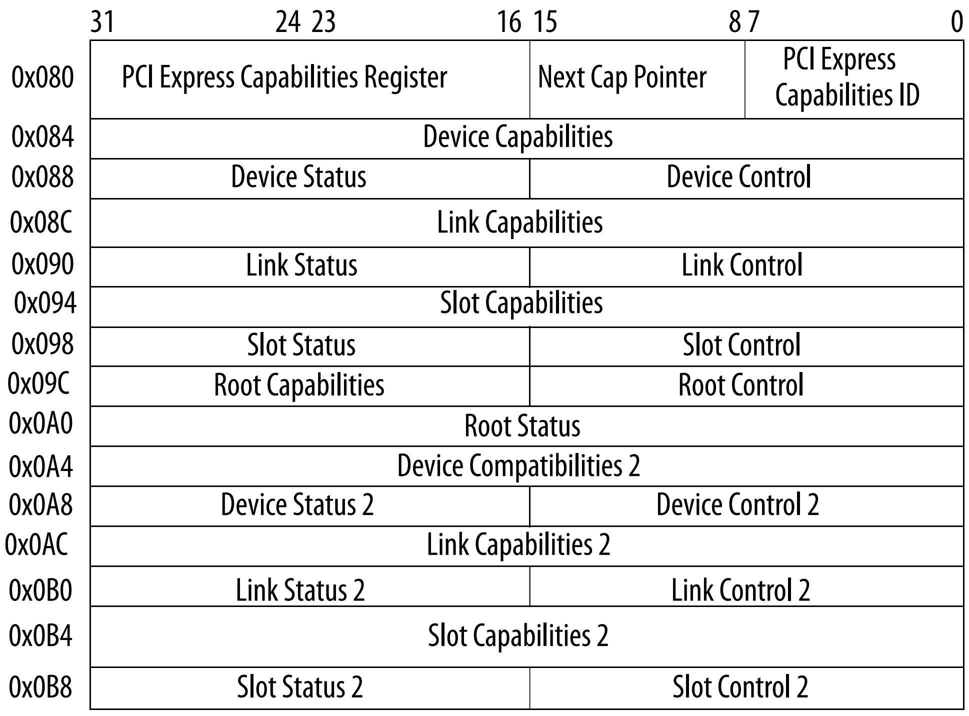
Figure 40. PCI Express AER Extended Capability Structure

Byte Offset	31:24	23:16	15:8	7:0
0x800	PCI Express Enhanced Capability Register			
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 Register			
0x830	Root Error Status Register			
0x834	Error Source Identification Register		Correctable Error Source Identification Register	

Note: Refer to the *Advanced Error Reporting Capability* section for more details about the PCI Express AER Extended Capability Structure.

Figure 41. PCI Express Capability Structure - Byte Address Offsets and Layout

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



	31 24	23 16	15 8	7 0
0x080	PCI Express Capabilities Register		Next Cap Pointer	PCI Express Capabilities 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 Compatibilities 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	

Related Information

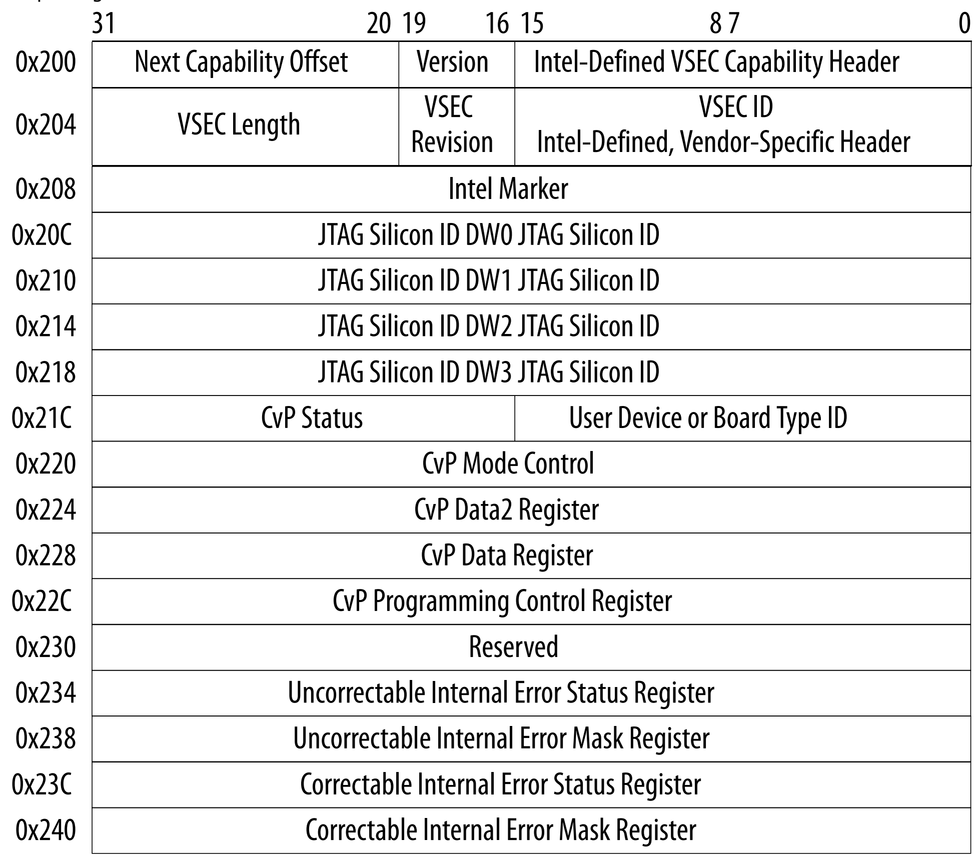
- PCI Express Base Specification 3.0
- PCI Local Bus Specification

Send Feedback

6.5. Intel-Defined VSEC Registers

Figure 42. VSEC Registers

This extended capability structure supports Configuration via Protocol (CvP) programming and detailed internal error reporting.



	31 20	19 16	15 8 7 0
0x200	Next Capability Offset	Version	Intel-Defined VSEC Capability Header
0x204	VSEC Length	VSEC Revision	VSEC ID Intel-Defined, Vendor-Specific Header
0x208	Intel 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		

Table 38. Intel-Defined VSEC Capability Register, 0x200

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

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

Table 39. Intel-Defined Vendor Specific Header

You can specify these values when you instantiate the Hard IP. These registers are read-only at run-time.

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

Table 40. Intel Marker Register

Bits	Register Description	Value	Access
[31:0]	Intel Marker. This read only register is an additional marker. If you use the standard Intel 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

Table 41. JTAG Silicon ID Register

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

Table 42. 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

6.6. CvP Registers

Table 43. CvP Status

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

Bits	Register Description	Reset Value	Access
[31:26]	Reserved	0x00	RO
[25]	PLD_CORE_READY. From FPGA fabric. This status bit is provided for debug.	Variable	RO
[24]	PLD_CLK_IN_USE. From clock switch module to fabric. This status bit is provided for debug.	Variable	RO
[23]	CVP_CONFIG_DONE. Indicates that the FPGA control block has completed the device configuration via CvP and there were no errors.	Variable	RO
[22]	Reserved	Variable	RO
[21]	USERMODE. Indicates if the configurable FPGA fabric is in user mode.	Variable	RO

continued...

Bits	Register Description	Reset Value	Access
[20]	CVP_EN. Indicates if the FPGA control block has enabled CvP mode.	Variable	RO
[19]	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
[18]	CVP_CONFIG_READY. Reflects the value of this signal from the FPGA control block, checked by software during programming algorithm.	Variable	RO
[17:0]	Reserved	Variable	RO

Table 44. CvP Mode Control

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

Bits	Register Description	Reset Value	Access
[31:16]	Reserved.	0x0000	RO
[15:8]	CVP_NUMCLKS. This is the number of clocks to send for every CvP data write. Set this field to one of the values below depending on your configuration image: • 0x01 for uncompressed and unencrypted images • 0x04 for uncompressed and encrypted images • 0x08 for all compressed images	0x00	RW
[7:3]	Reserved.	0x0	RO
[2]	CVP_FULLCONFIG. Request that the FPGA control block reconfigure the entire FPGA including the Intel Arria 10 or Intel Cyclone 10 GX 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 IP core 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 routed to the FPGA fabric.	1'b0	RW

Table 45. CvP Data 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.

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

Table 46. CvP Programming Control Register

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

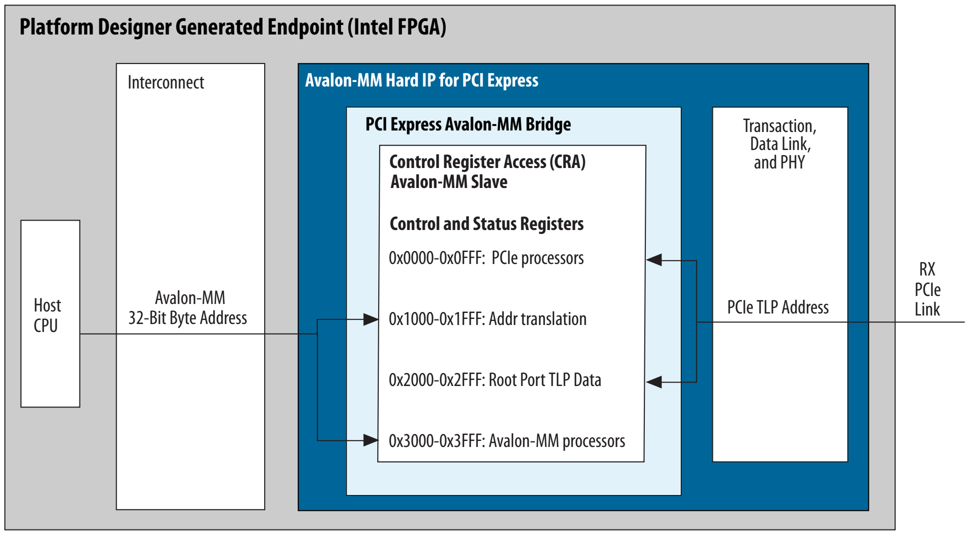
Bits	Register Description	Reset Value	Access
[31:2]	Reserved.	0x0000	RO
[1]	START_XFER. Sets the CvP output to the FPGA control block indicating the start of a transfer.	1’b0	RW
[0]	CVP_CONFIG. When asserted, instructs that the FPGA control block begin a transfer via CvP.	1’b0	RW

6.7. 64- or 128-Bit Avalon-MM Bridge Register Descriptions

The CRA Avalon-MM slave module provides access control and status registers in the PCI Express Avalon-MM bridge. In addition, it provides access to selected Configuration Space registers and link status registers in read-only mode. This module is optional. However, you must include it to access the registers.

The control and status register address space is 16 KB. Each 4 KB 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 Intel Arria 10 or Intel Cyclone 10 GX 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 43. Accesses to the Avalon-MM Bridge Control and Status Register


Platform Designer Generated Endpoint (Intel FPGA)
Interconnect
Avalon-MM Hard IP for PCI Express
PCI Express Avalon-MM Bridge
Control Register Access (CRA) Avalon-MM Slave
Control and Status Registers
0x0000-0x0FFF: PCIe processors
0x1000-0x1FFF: Addr translation
0x2000-0x2FFF: Root Port TLP Data
0x3000-0x3FFF: Avalon-MM processors
Transaction, Data Link, and PHY
PCIe TLP Address
RX PCIe Link
Host CPU
Avalon-MM 32-Bit Byte Address

The following table describes the four subregions.

Table 47. Avalon-MM Control and Status Register Address Spaces

Address Range	Address Space Usage
0x0000-0x0FFF	Registers typically intended for access by PCI Express link partner 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 the PCI Express link partner, 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 for Configuration TLPs, I/O TLPs, single dword Memory Read and Write requests, and receive interrupts from an Endpoint.
0x3000-0x3FFF	Registers typically intended for access by Avalon-MM processors only. Provides host access to selected Configuration Space and status 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 48. 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
0x0800–0x081F	PCI Express-to-Avalon-MM Mailbox Registers
0x0900–x091F	Avalon-MM to PCI Express Mailbox Registers

continued...

Address Range	Register
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
0x3A00-0x3A1F	Avalon-MM to PCI Express Mailbox Registers
0x3B00-0x3B1F	PCI Express to Avalon-MM Mailbox Registers
0x3C00-0x3C6C	Host (Avalon-MM master) access to selected Configuration Space and status registers.

6.7.1. Avalon-MM to PCI Express Interrupt Registers

6.7.1.1. Avalon-MM to PCI Express Interrupt Status Registers

These registers contain the status of various signals in the PCI Express Avalon-MM bridge logic. These registers allow MSI or legacy 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 49. Avalon-MM to PCI Express Interrupt Status Register, 0x0060

Bit	Name	Access	Description
[31:24]	Reserved	N/A	N/A
[23]	A2P_MAILBOX_INT7	RW1C	Set to 1 when the A2P_MAILBOX7 register is written to
[22]	A2P_MAILBOX_INT6	RW1C	1 when the A2P_MAILBOX6 register is written to
[21]	A2P_MAILBOX_INT5	RW1C	Set 10 1 when the A2P_MAILBOX5 register is written to
[20]	A2P_MAILBOX_INT4	RW1C	Set 10 1 when the A2P_MAILBOX4 register is written to
[19]	A2P_MAILBOX_INT3	RW1C	Set 10 1 when the A2P_MAILBOX3 register is written to
[18]	A2P_MAILBOX_INT2	RW1C	Set 10 1 when the A2P_MAILBOX2 register is written to
[17]	A2P_MAILBOX_INT1	RW1C	Set 10 1 when the A2P_MAILBOX1 register is written to
[16]	A2P_MAILBOX_INT0	RW1C	Set 10 1 when the A2P_MAILBOX0 register 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.

continued...

Bit	Name	Access	Description
			A PCIe variant may have as many as 16 distinct IRQ input ports. Each AVL_IRQ_ASSERTED[] bit reflects the value on the corresponding IRQ input port.

6.7.1.2. Avalon-MM to PCI Express Interrupt Enable Registers

The interrupt enable registers enable either MSI or legacy interrupts.

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.

Table 50. Avalon-MM to PCI Express Interrupt Enable Register, 0x0050

Bits	Name	Access	Description
[31:24]	Reserved	N/A	N/A
[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.
[15:0]	`AVL_IRQ[15:0]`	RW	Enables generation of PCI Express interrupts when a specified Avalon-MM interrupt signal is asserted. Your system may have as many as 16 individual input interrupt signals.

Table 51. Avalon-MM Interrupt Vector Register - 0x0060

Bits	Name	Access	Description
[31:16]	Reserved	N/A	N/A
[15:0]	`AVL_IRQ_Vector`	RO	Stores the interrupt vector of the system interconnect fabric. When the host receives an interrupt, it should read this register to determine the servicing priority.

6.7.1.3. 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 52. PCI Express to Avalon-MM Mailbox Registers, 0x0800–0x081F

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

continued...

Address	Name	Access	Description
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 53. Avalon-MM to PCI Express Mailbox Registers, 0x0900–0x091F

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

6.7.1.4. 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_SPACE511 and A2P_ADDR_MAP_LO511 and 0x1FFC contains A2P_ADDR_MAP_HI511.

Table 54. 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. 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_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.
0x1008	[1:0]	A2P_ADDR_SPACE1	RW	Address space indication for entry 1. This entry is available only if the number of translation table entries (**Number of address pages**) is greater than 1. The same encodings are defined for A2P_ADDR_SPACE1 as for A2P_ADDR_SPACE0.:
	[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.

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

These registers record the status of various signals in the PCI Express Avalon-MM bridge logic. They allow Avalon-MM 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, nothing in the hardware prevents a PCI Express Avalon-MM bridge master port from accessing these registers.

The following table describes the Interrupt Status register for Endpoints. It records the status of all conditions that can cause an Avalon-MM interrupt to be asserted.

Table 55. 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.
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.

continued...

Bits	Name	Access	Description
2	TX_FIFO_EMPTY	RW1C	When set to 1, indicates that the TX buffer is empty. Application Layer logic can read this bit to determine if all of the TX buffer is empty before safely changing the translation address entries. This bit is available only for Legacy Endpoints.
[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. Typically, a single process in either the PCI Express or Avalon-MM domain handles the condition reported by the interrupt.

Table 56. Avalon-MM Interrupt Enable Register, 0x3070

Bits	Name	Access	Description
[31:24]	Reserved	N/A	Reserved
[23]	P2A_MAILBOX_INT7	RW1C	Set to a 1 when the P2A_MAILBOX7 is written to.
[22]	P2A_MAILBOX_INT6		Set to a 1 when the P2A_MAILBOX6
[21]	P2A_MAILBOX_INT5		Set to a 1 when the P2A_MAILBOX5
[20]	P2A_MAILBOX_INT4		Set to a 1 when the P2A_MAILBOX4
[19]	P2A_MAILBOX_INT3		Set to a 1 when the P2A_MAILBOX3
[18]	P2A_MAILBOX_INT2		Set to a 1 when the P2A_MAILBOX2
[17]	P2A_MAILBOX_INT1		Set to a 1 when the P2A_MAILBOX1
[16]	P2A_MAILBOX_INT0		Set to a 1 when the P2A_MAILBOX0
[15:0]	Reserved	N/A	Reserved

6.7.1.6. 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 57. 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
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 58. 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

6.7.1.7. Control Register Access (CRA) Avalon-MM Slave Port

Table 59. Configuration Space Register Descriptions

For registers that are less than 32 bits, the upper bits are unused.

Byte Offset	Register	Dir	Description
14'h3C00	`cfg_dev_ctrl[15:0]`	O	`cfg_devctrl[15:0]` is device control for the PCI Express capability structure.
14'h3C04	`cfg_dev_ctrl2[15:0]`	O	`cfg_dev2ctrl[15:0]` is device control 2 for the PCI Express capability structure.
14'h3C08	`cfg_link_ctrl[15:0]`	O	`cfg_link_ctrl[15:0]`is the primary Link Control of the PCI Express capability structure. For Gen2 or Gen3 operation, you must write a 1’b1 to Retrain Link bit (Bit[5] of the `cfg_link_ctrl`) of the Root Port to initiate retraining to a higher data rate after the initial link training to Gen1 L0 state. Retraining directs the LTSSM to the Recovery state. Retraining to a higher data rate is not automatic for the Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express IP Core even if both devices on the link are capable of a higher data rate.
14'h3C0C	`cfg_link_ctrl2[15:0]`	O	`cfg_link_ctrl2[31:16]` is the secondary Link Control register of the PCI Express capability structure for Gen2 operation. For Gen1 variants, the link bandwidth notification bit is always set to 0. For Gen2 variants, this bit is set to 1.
14'h3C10	`cfg_prm_cmd[15:0]`	O	Base/Primary Command register for the PCI Configuration Space.
14'h3C14	`cfg_root_ctrl[7:0]`	O	Root control and status register of the PCI-Express capability. This register is only available in Root Port mode.
14'h3C18	`cfg_sec_ctrl[15:0]`	O	Secondary bus Control and Status register of the PCI-Express capability. This register is only available in Root Port mode.
14'h3C1C	`cfg_secbus[7:0]`	O	Secondary bus number. Available in Root Port mode.
14'h3C20	`cfg_subbus[7:0]`	O	Subordinate bus number. Available in Root Port mode.
14'h3C24	`cfg_msi_addr_low[31:0]`	O	`cfg_msi_add[31:0]` is the MSI message address.
14'h3C28	`cfg_msi_addr_hi[63:32]`	O	`cfg_msi_add[63:32]` is the MSI upper message address.
14'h3C2C	`cfg_io_bas[19:0]`	O	The IO base register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h3C30	`cfg_io_lim[19:0]`	O	The IO limit register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h3C34	`cfg_np_bas[11:0]`	O	The non-prefetchable memory base register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h3C38	`cfg_np_lim[11:0]`	O	The non-prefetchable memory limit register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h3C3C	`cfg_pr_bas_low[31:0]`	O	The lower 32 bits of the prefetchable base register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h3C40	`cfg_pr_bas_hi[43:32]`	O	The upper 12 bits of the prefetchable base registers of the Type1 Configuration Space. This register is only available in Root Port mode.

continued...

Byte Offset	Register	Dir	Description
14'h3C44	`cfg_pr_lim_low[31:0]`	O	The lower 32 bits of the prefetchable limit registers of the Type1 Configuration Space. Available in Root Port mode.
14'h3C48	`cfg_pr_lim_hi[43:32]`	O	The upper 12 bits of the prefetchable limit registers of the Type1 Configuration Space. Available in Root Port mode.
14'h3C4C	`cfg_pmcsr[31:0]`	O	`cfg_pmcsr[31:16]` is Power Management Control and `cfg_pmcsr[15:0]`is the Power Management Status register.
14'h3C50	`cfg_msixcsr[15:0]`	O	MSI-X message control register.
14'h3C54	`cfg_msicsr[15:0]`	O	MSI message control.
14'h3C58	`cfg_tcvcmap[23:0]`	O	Configuration traffic class (TC)/virtual channel (VC) mapping. The Application Layer uses this signal to generate a TLP mapped to the appropriate channel based on the traffic class of the packet. The following encodings are defined: • `cfg_tcvcmap[2:0]`: Mapping for TC0 (always 0). • `cfg_tcvcmap[5:3]`: Mapping for TC1. • `cfg_tcvcmap[8:6]`: Mapping for TC2. • `cfg_tcvcmap[11:9]`: Mapping for TC3. • `cfg_tcvcmap[14:12]`: Mapping for TC4. • `cfg_tcvcmap[17:15]`: Mapping for TC5. • `cfg_tcvcmap[20:18]`: Mapping for TC6. • `cfg_tcvcmap[23:21]`: Mapping for TC7.
14'h3C5C	`cfg_msi_data[15:0]`	O	`cfg_msi_data[15:0]` is message data for MSI.
14'h3C60	`cfg_busdev[12:0]`	O	Bus/Device Number captured by or programmed in the Hard IP.
14'h3C64	`ltssm_reg[4:0]`	O	Specifies the current LTSSM state. The LTSSM state machine encoding defines the following states: • 00000: Detect.Quiet • 00001: Detect.Active • 00010: Polling.Active • 00011: Polling.Compliance • 00100: Polling.Configuration • 00101: Polling.Speed • 00110: config.Linkwidthstart • 00111: Config.Linkaccept • 01000: Config.Lanenumaccept • 01001: Config.Lanenumwait • 01010: Config.Complete • 01011: Config.Idle • 01100: Recovery.Rcvlock • 01101: Recovery.Rcvconfig • 01110: Recovery.Idle • 01111: L0 • 10000: Disable • 10001: Loopback.Entry • 10010: Loopback.Active • 10011: Loopback.Exit • 10100: Hot.Reset • 10101: LOs • 11001: L2.transmit.Wake • 11010: Speed.Recovery

continued...

Byte Offset	Register	Dir	Description
			• 11011: Recovery.Equalization, Phase 0 • 11100: Recovery.Equalization, Phase 1 • 11101: Recovery.Equalization, Phase 2 • 11110: recovery.Equalization, Phase 3
14'h3C68	`current_speed_reg[1:0]`	O	Indicates the current speed of the PCIe link. The following encodings are defined: • 2b'00: Undefined • 2b'01: Gen1 • 2b'10: Gen2 • 2b'11: Gen3
14'h3C6C	`lane_act_reg[3:0]`	O	Lane Active Mode: This signal indicates the number of lanes that configured during link training. The following encodings are defined: • 4'b0001: 1 lane • 4'b0010: 2 lanes • 4'b0100: 4 lanes • 4'b1000: 8 lanes

6.8. 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, Message TLPs with data payload, 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 request to downstream ports.

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.

6.8.1. 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 at address 0x2000.
2. Write the next 32 bits of the TX TLP to RP_TX_REG1 at address 0x2004.
3. Write the RP_TX_CNTRL.SOP to 1'b1 (RP_TX_CNTRL is at address 0x2008) 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 to 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.

6.8.2. Sending a Read TLP or Receiving a Non-Posted Completion TLP

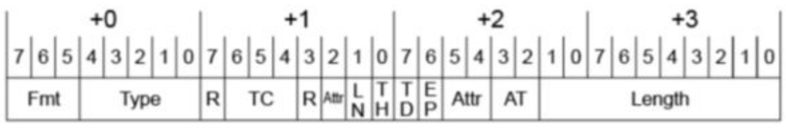
The TLPs associated with the Non-Posted TX requests are stored in the RP_RX_CPL FIFO buffer and subsequently loaded into RP_RXCPL registers. The Application Layer performs the following sequence to retrieve the TLP.

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 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 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.

6.8.3. Examples of Reading and Writing BAR0 Using the CRA Interface

You can use the CRA interface to send TLP requests. The Fmt and Type fields of the TLP Header provide the information required to determine the size of the remaining part of the TLP Header, and if the packet contains a data payload following the Header.

Figure 44. TLP Header Format



The CRA interface uses register addresses 0x2000, 0x2004, and 0x2008 to send TLPs, and register addresses 0x2010, 0x2014, and 0x2018 to check Completions. For details about these registers, refer to the table *Root Port TLP Data Registers, 0x2000 - 0x2FFF*. Below are examples of how to use Type 0 configuration TLPs to read from BAR0 and write to it.

1. Use the CRA interface to read an uninitialized BAR0 using a Type 0 configuration TLP with the format as shown below:

```
| fmt |   typ   |t|  tc |t|a|l|t|t|e|att| at|       length       |
| 000b|  00100b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 001 _______|
|________ req_id: 0000 _________|___ tag: 17 ___|lbe: 0 |fbe: f |
|    bdf.bus   |bdf.dev|bdf.func|rsvd20|reg_no.ext|reg_no.low|rsv|
|______ 01 ___|__ 00 _|___ 0 __|_ 0 __|___ 0 ____|___ 04 ___| 0 |
04000001 0000170f 01000010
```

To send the TLP using the CRA interface, do the following steps:

a. Write 0x0400_0001 to CRA interface address 0x2000.

b. Write 0x0000_170F to CRA interface address 0x2004.

c. Write 0x0000_0001 to CRA interface address 0x2008 (Start of Packet).

d. Write 0x0100_0010 to CRA interface address 0x2000.

e. Write 0x0000_0000 to CRA interface address 0x2004.

f. Write 0x0000_0002 to CRA interface address 0x2008 (End of Packet).

Check the corresponding Completion using the CRA interface. The Completion TLP has four dwords, with the first three dwords as shown below, followed by one dword of uninitialized BAR0 value (which is 0xFFEF0010 in the following picture).

```
| fmt |   typ   |t|  tc |t|a|l|t|t|e|att| at|       length      |
| 010b|  01010b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 001 _______|
|_____ cpl_id: 0100 ___|cpl_status: 0|bcm: 0|__ byte_cnt: 004 __|
|_____ req_id: 0000 ___|___ tag: 17 ___|rsvd20: 0| low_addr: 00 |
4a000001 01000004 00001700 ffef0010
```

To read the Completion using the CRA interface, do the following steps:

a. Keep reading CRA interface address 0x2010 until bit [0] = 0x1 (indicating the Completion packet has arrived, and you can receive the SOP in the next step.

b. Read CRA interface address 0x2014. The read data value is 0x4A00_0001.

c. Read CRA interface address 0x2018. The read data value is 0x0100_0004.

d. Read CRA interface address 0x2010. In this example, bits [1:0] = 0x2 (indicating that you will receive the EOP in the next step). If bits [1:0] = 0x0, the values read in the next two steps are still in the middle of the packet. In this case, you need to keep reading 0x2010, 0x2014, and 0x2018 after performing the next two steps.

e. Read CRA interface address 0x2014. The read data value is 0x0000_1700.

f. Read CRA interface address 0x2018. The read data value is BAR0's uninitialized value.

2. Use the CRA interface to initialize BAR0 with 0xFFFF_FFFF using a Type 0 configuration TLP with the format as shown below:

```
| fmt |   typ   |t|  tc |t|a|l|t|t|e|att| at|       length      |
| 010b|  00100b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 001 _______|
|________ req_id: 0000 _________|___ tag: 11 ___|lbe: 0 |fbe: f |
|    bdf.bus  |bdf.dev|bdf.func|rsvd20|reg_no.ext|reg_no.low|rsv|
|______ 01 ___|__ 00 _|___ 0 __|_ 0 __|___ 0 ____|___ 04 ___| 0 |
44000001 0000110f 01000010 ffffffff
```

To send the TLP using the CRA interface, do the following steps:

a. Write 0x4400_0001 to CRA interface address 0x2000.

b. Write 0x0000_110F to CRA interface address 0x2004.

c. Write 0x0000_0001 to CRA interface address 0x2008 (Start of Packet).

d. Write 0x0100_0010 to CRA interface address 0x2000.

e. Write 0xFFFF_FFFF to CRA interface address 0x2004.

f. Write 0x0000_0002 to CRA interface address 0x2008 (End of Packet).

Check the corresponding Completion using the CRA interface. The Completion TLP has three dwords as shown below:

```
| fmt |   typ   |t|  tc |t|a|l|t|t|e|att| at|       length      |
| 000b|  01010b |0|_ 0 _|0|0|0|0|0|0| 0 | 0 |_______ 000 _______|
|_____ cpl_id: 0100 ___|cpl_status: 0|bcm: 0|__ byte_cnt: 004 __|
|_____ req_id: 0000 ___|___ tag: 11 ___|rsvd20: 0| low_addr: 00 |
0a000000 01000004 00001100
```

To read the Completion using the CRA interface, do the following steps:

a. Keep reading CRA interface address 0x2010 until bit [0] = 0x1 (indicating the Completion packet has arrived, and you can receive the SOP in the next step).

b. Read CRA interface address 0x2014. The read data value is 0x0A00_0000.

c. Read CRA interface address 0x2018. The read data value is 0x0100_0004.

d. Read CRA interface address 0x2010. In this example, bits [1:0] = 0x2.

e. Read CRA interface address 0x2014. The read data value is 0x0000_1100.

You can repeat Step 1 to read BAR0 after writing 0xFFFF_FFFF to it, and repeat Step 2 to configure the BAR0 address space.

Use the same method to configure BAR1, BAR2, BAR3, BAR4 and BAR5.

6.8.4. 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 60. Avalon-MM Interrupt Status Registers for Root Ports, 0x3060

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

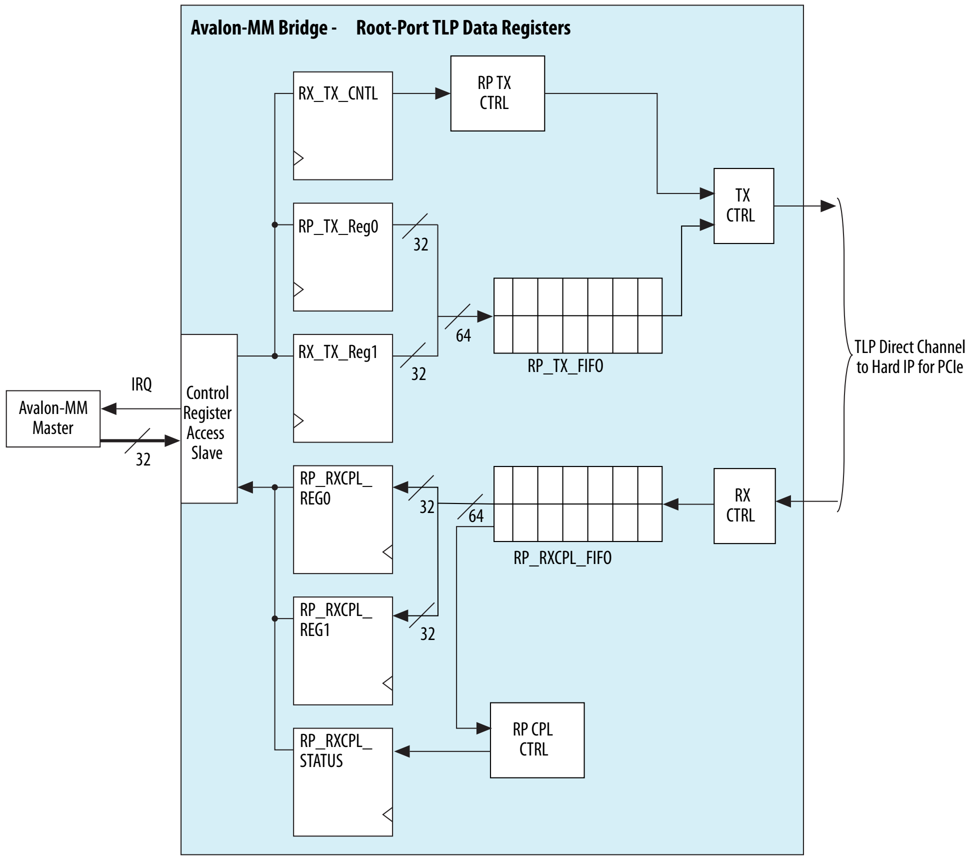
Table 61. INT-X Interrupt Enable Register for Root Ports, 0x3070

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.
[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.

6.8.5. 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 45. 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 62. 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. *Note:* Both bits [1] and [0] are equal to 0 for all cycles in the packet except for the SOP and EOP cycles.
0x2010	[31:2]	Reserved	—	—

continued...

Root-Port Request Registers				Address Range: 0x2800-0x2018
Address	**Bits**	**Name**	**Access**	**Description**
	[1]	RP_RXCPL_STATUS.EOP	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.
	[0]	RP_RXCPL_STATUS.SOP	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.
0x2014	[31:0]	RP_RXCPL_REG0	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.

Related Information

Avalon-MM Bridge TLPs on page 167

6.9. Uncorrectable Internal Error Mask Register

Table 63. Uncorrectable Internal Error Mask Register

The `Uncorrectable Internal Error Mask` 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. The access code *RWS* stands for Read Write Sticky meaning the value is retained after a soft reset of the IP core.

Bits	Register Description	Reset Value	Access
[31:12]	Reserved.	1b'0	RO
[11]	Mask for RX buffer posted and completion overflow error.	1b'0	RWS
[10]	Reserved	1b'1	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'1	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

6.10. Uncorrectable Internal Error Status Register

Table 64. 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 custom logic. The access code RW1CS represents Read Write 1 to Clear Sticky.

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

Related Information

PCI Express Base Specification 3.0

6.11. Correctable Internal Error Mask Register

Table 65. 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.

Bits	Register Description	Reset Value	Access
[31:8]	Reserved.	0	RO
[7]	Reserved.	1	RO
[6]	Mask for Corrected Internal Error reported by the Application Layer.	1	RWS
[5]	Mask for configuration error detected in CvP mode.	1	RWS

continued...

Bits	Register Description	Reset Value	Access
[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

6.12. Correctable Internal Error Status Register

Table 66. 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. Only use this register to observe behavior, not to drive logic custom logic.

Bits	Register Description	Reset Value	Access
[31:7]	Reserved.	0	RO
[6]	Corrected Internal Error reported by the Application Layer.	0	RW1CS
[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

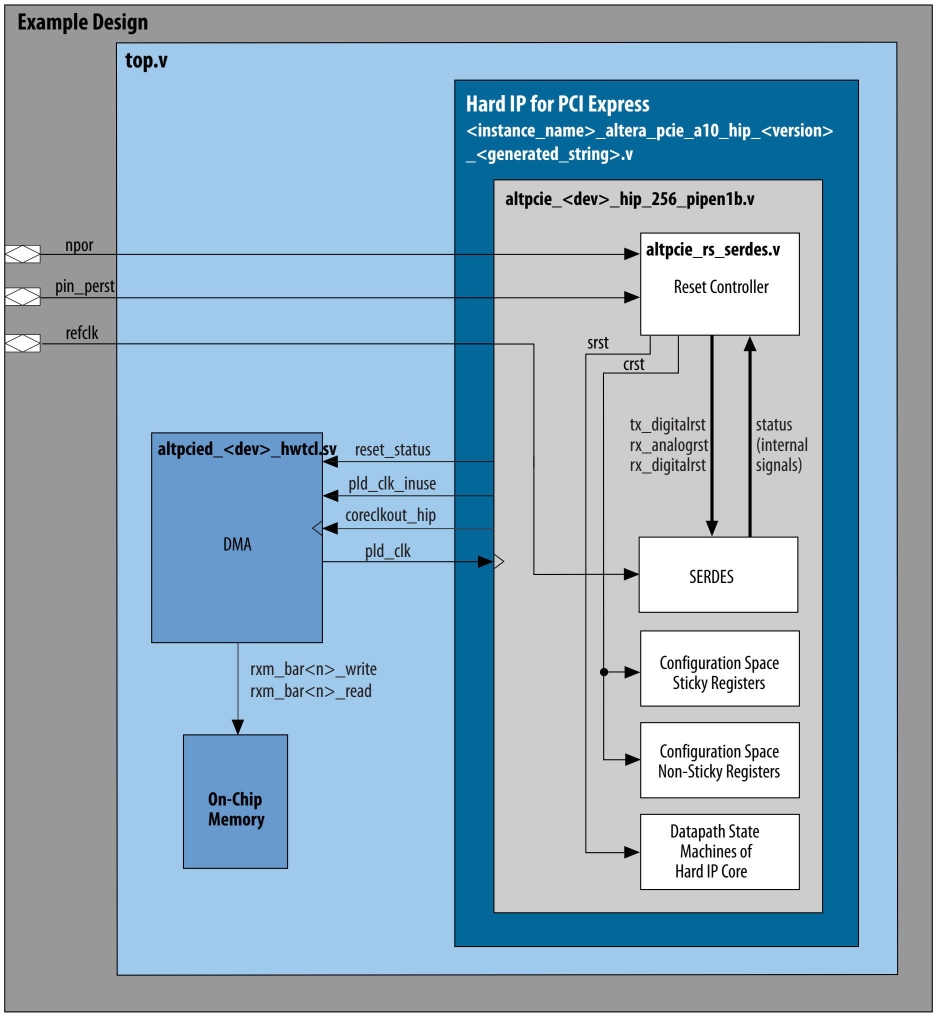
Related Information

PCI Express Base Specification 3.0

7. Reset and Clocks

The following figure shows the hard reset controller that is embedded inside the Hard IP for PCI Express. This controller takes in the `npor` and `pin_perst` inputs and generates the internal reset signals for other modules in the Hard IP.

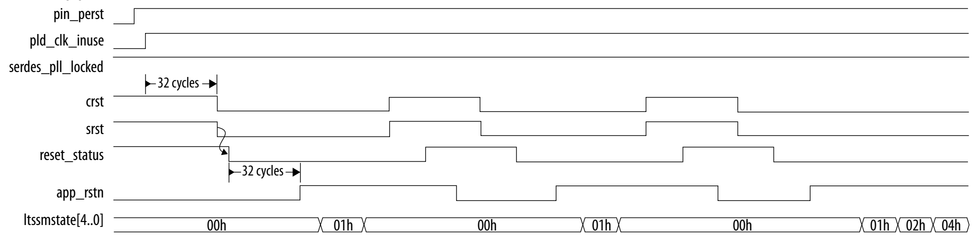
Figure 46. Reset Controller in Intel Arria 10 or Intel Cyclone 10 GX Devices



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

Figure 47. Hard IP for PCI Express and Application Logic Reset Sequence

Your Application Layer can instantiate a module with logic that implements the timing diagram shown below to generate `app_rstn`, which resets the Application Layer logic.

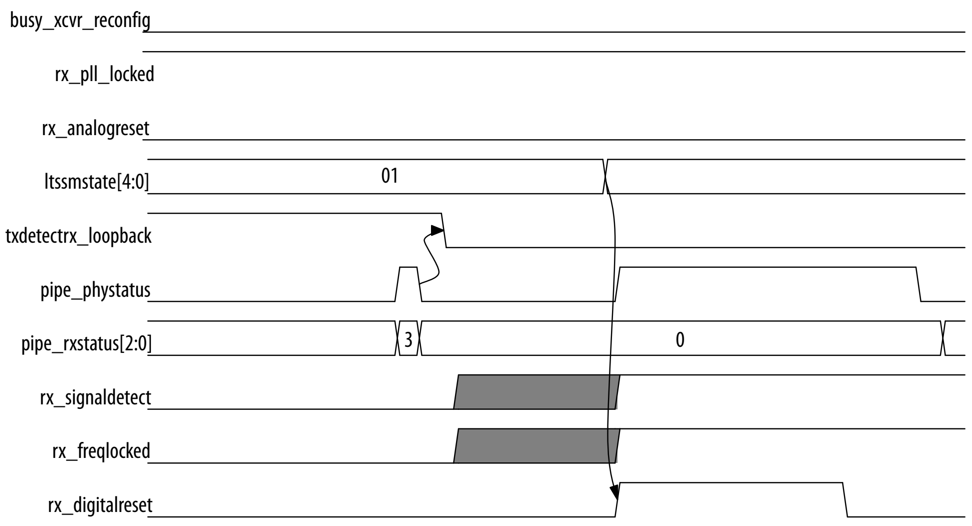


This reset sequence includes the following steps:

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 `pld_clk`cycles after `reset_status` is released.

   Note: `reset_status` may be toggling until the host and its receivers are detected during the link training sequence (`ltssmstate[4:0]` = 0x02).

Figure 48. RX Transceiver Reset Sequence

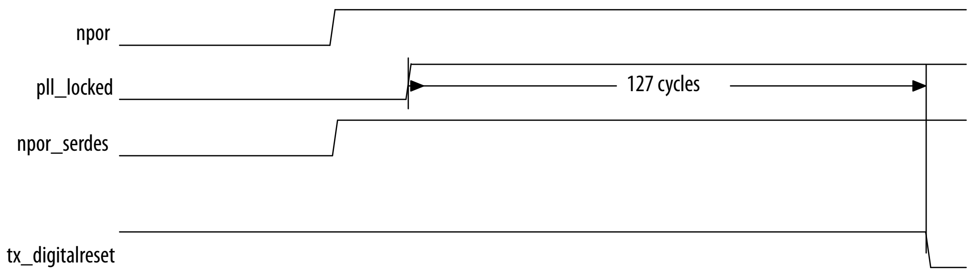


Send Feedback

The RX transceiver reset sequence 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 49. TX Transceiver Reset Sequence



The TX transceiver reset sequence includes the following steps:

1. After `npor` is deasserted, the IP 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 `pld_clk` cycles before deasserting `tx_digitalreset`.

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

7.2. Clocks

The Hard IP contains a clock domain crossing (CDC) synchronizer at the interface between the PHY/MAC and the DLL layers. The synchronizer 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.

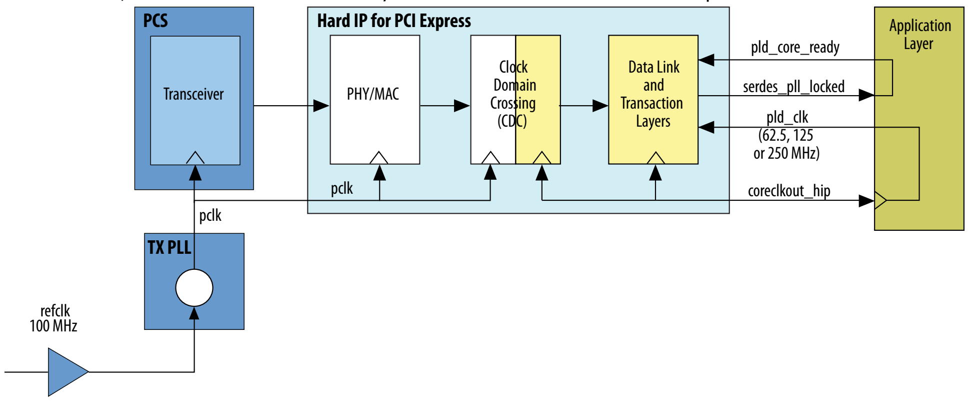
Related Information

PCI Express Base Specification 3.0

7.2.1. Clock Domains

Figure 50. Clock Domains and Clock Generation for the Application Layer

The following illustrates the clock domains when using `coreclkout_hip` to drive the Application Layer and the `pld_clk` of the IP core. The Intel-provided example design connects `coreclkout_hip` to the `pld_clk`. However, this connection is not mandatory. Inside the Hard IP for PCI Express, the blocks shown in white are in the `pclk` domain, while the blocks shown in yellow are in the `coreclkout_hip` domain.



As this figure indicates, the IP core includes the following clock domains: `pclk`, `coreclkout_hip` and `pld_clk`.

7.2.1.1. coreclkout_hip

Table 67. Application Layer Clock Frequency for All Combinations of Link Width, Data Rate and Application Layer Interface Widths

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 Application Layer to Transaction Layer interface. 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.)

Link Width	Maximum Link Rate	Avalon Interface Width	coreclkout_hip
×1	Gen1	64	62.5 MHz[6]
×1	Gen1	64	125 MHz
×2	Gen1	64	125 MHz
×4	Gen1	64	125 MHz
×2	Gen2	64	125 MHz
×4	Gen2	128	125 MHz
×1	Gen1	64	62.5 MHz[7]
×1	Gen1	64	125 MHz
×2	Gen1	64	125 MHz
×4	Gen1	64	125 MHz
×8	Gen1	64	250 MHz

continued...

[6] This mode saves power

[7] This mode saves power

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

7.2.1.2. pld_clk

`coreclkout_hip` can drive the Application Layer clock along with the `pld_clk` input to the 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.

7.2.2. Clock Summary

Table 68. Clock Summary

Name	Frequency	Clock Domain
`coreclkout_hip`	62.5, 125 or 250 MHz	Avalon-ST interface between the Transaction and Application Layers.
`pld_clk`	`pld_clk` has a maximum frequency of 250 MHz and a minimum frequency that can be equal or more than the `coreclkout_hip` frequency, depending on the link width, link rate, and Avalon interface width as indicated in the table for the Application Layer clock frequency above.	Application and Transaction Layers.
`refclk`	100 MHz	SERDES (transceiver). Dedicated free running input clock to the SERDES block.

8. Interrupts for Endpoints

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.

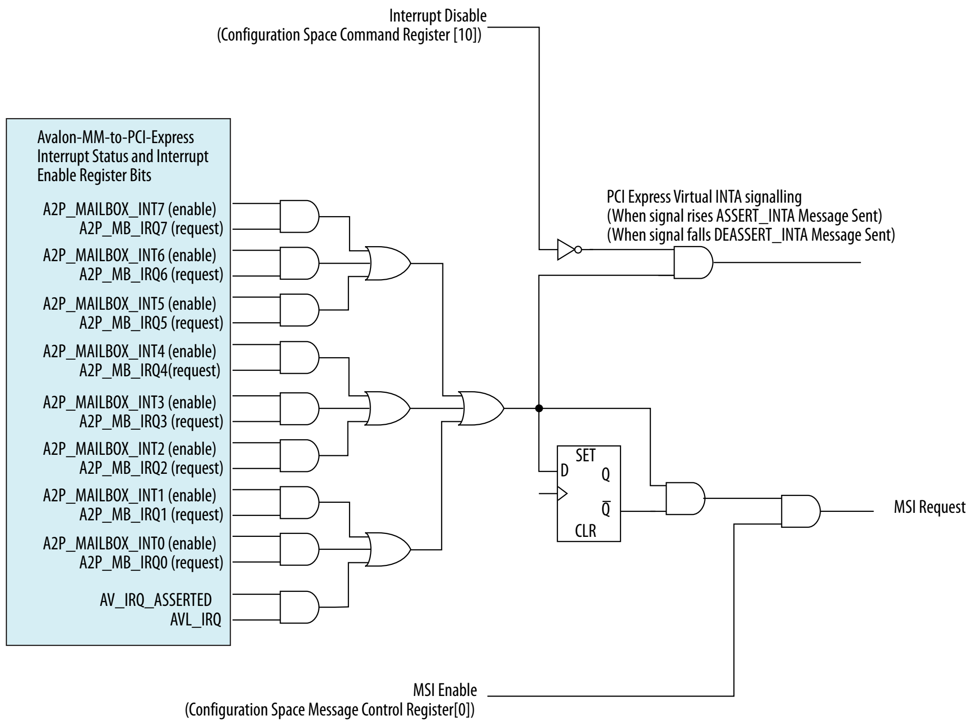
The PCI Express Avalon-MM bridge supports the Avalon-MM individual requests interrupt scheme: multiple input signals indicate incoming interrupt requests, and software must determine priorities for servicing simultaneous interrupts.

The RX master module port has up to 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` 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` mailbox writes, the status bits should be cleared in the `Avalon-MM to PCI Express Interrupt Status Register`. 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.

Figure 51. Avalon-MM Interrupt Propagation to the PCI Express Link


Interrupt Disable
(Configuration Space Command Register [10])
Avalon-MM-to-PCI-Express Interrupt Status and Interrupt Enable Register Bits
A2P_MAILBOX_INT7 (enable)
A2P_MB_IRQ7 (request)
A2P_MAILBOX_INT6 (enable)
A2P_MB_IRQ6 (request)
A2P_MAILBOX_INT5 (enable)
A2P_MB_IRQ5 (request)
A2P_MAILBOX_INT4 (enable)
A2P_MB_IRQ4(request)
A2P_MAILBOX_INT3 (enable)
A2P_MB_IRQ3 (request)
A2P_MAILBOX_INT2 (enable)
A2P_MB_IRQ2 (request)
A2P_MAILBOX_INT1 (enable)
A2P_MB_IRQ1 (request)
A2P_MAILBOX_INT0 (enable)
A2P_MB_IRQ0 (request)
AV_IRQ_ASSERTED
AVL_IRQ
PCI Express Virtual INTA signalling
(When signal rises ASSERT_INTA Message Sent)
(When signal falls DEASSERT_INTA Message Sent)
SET
D
Q
Q̄
CLR
MSI Request
MSI Enable
(Configuration Space Message Control Register[0])

Related Information

- Avalon-MM to PCI Express Interrupt Enable Registers on page 77
- Avalon-MM to PCI Express Interrupt Status Registers on page 76

8.1. 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.

8.2. Generation of Avalon-MM Interrupts

The 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` 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` 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

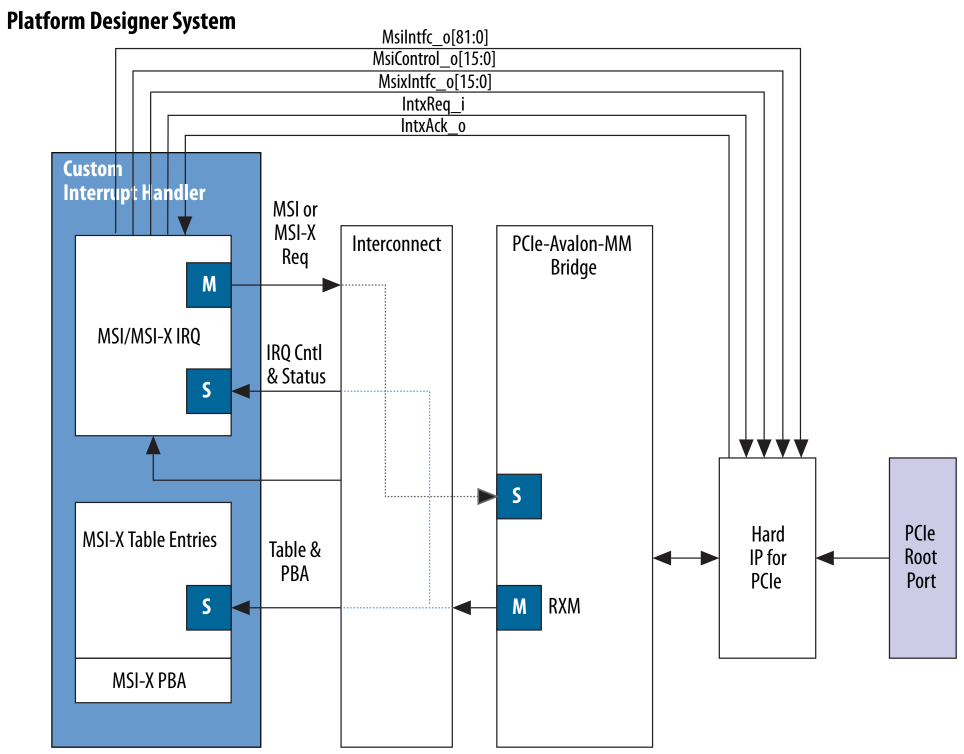
- Avalon-MM to PCI Express Interrupt Status Registers on page 76
- PCI Express to Avalon-MM Interrupt Status and Enable Registers for Endpoints on page 79

8.3. Interrupts for Endpoints 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 parameter editor, 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 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 the PCIe Root Port.
- An MSI-X table to store the MSI-X table entries. The PCIe Root Port sets up this table.

Figure 52. 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*.

For more information about implementing interrupts, including an MSI design example, refer to *Handling PCIe Interrupts* on the Intel FPGA wiki.

Related Information

- Interrupts for Endpoints on page 98
- PCI Local Bus Specification, Revision 2.3
- Handling PCIe Interrupts

9. Error Handling

Each PCI Express compliant device must implement a basic level of error management and can optionally implement advanced error management. The IP core implements both basic and advanced error reporting. Error handling for a Root Port is more complex than that of an Endpoint.

Table 69. Error Classification

The *PCI Express Base Specification* defines three types of errors, outlined in the following table.

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

9.1. Physical Layer Errors

Table 70. Errors Detected by the Physical Layer

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

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: — 3'b100: 8B/10B Decode Error — 3'b101: Elastic Buffer Overflow — 3'b110: Elastic Buffer Underflow — 3'b111: Disparity Error • Deskew error caused by overflow of the multilane deskew FIFO. • Control symbol received in wrong lane.

9.2. Data Link Layer Errors

Table 71. 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.

9.3. Transaction Layer Errors

Table 72. 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 non-posted request, the Hard IP block generates a completion with Unsupported Request status.

continued...

Error	Type	Description
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 the address range defined by the Base and Limit Address registers
Completion timeout	Uncorrectable (non-fatal)	This error occurs when a request originating from the Application Layer does not generate a corresponding completion TLP within the established time. It is the responsibility of the Application Layer logic to provide the completion timeout mechanism. The completion timeout should be reported from the Transaction Layer using the `cpl_err[0]` signal.
Completer abort [(1)]	Uncorrectable (non-fatal)	The Application Layer reports this error using the `cpl_err[2]`signal when it aborts receipt of a TLP.
Unexpected completion	Uncorrectable (non-fatal)	This error is caused by an unexpected completion transaction. The Hard IP block handles the following conditions: • The Requester ID in the completion packet does not match the Configured ID of the Endpoint. • The completion packet has an invalid tag number. (Typically, the tag used in the completion packet exceeds the number of tags specified.) • The completion packet has a tag that does not match an outstanding request. • The completion packet for a request that was to I/O or Configuration Space has a length greater than 1 dword. • The completion status is Configuration Retry Status (CRS) in response to a request that was not to Configuration Space. In all of the above cases, the TLP is not presented to the Application Layer; the Hard IP block deletes it. The Application Layer can detect and report other unexpected completion conditions using the `cpl_err[2]` signal. For example, the Application Layer can report cases where the total length of the received successful completions do not match the original read request length.
Receiver overflow [(1)]	Uncorrectable (fatal)	This error occurs when a component receives a TLP that violates the FC credits allocated for this type of TLP. In all cases the hard IP block deletes the TLP and it is not presented to the Application Layer.
Flow control protocol error (FCPE) [(1)]	Uncorrectable (fatal)	This error occurs when a component does not receive update flow control credits with the 200 µs limit.
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.

continued...

Error	Type	Description
		• 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 KB 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*.		

9.4. 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.

Table 73. 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

9.5. Uncorrectable and Correctable Error Status Bits

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

Figure 53. 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.

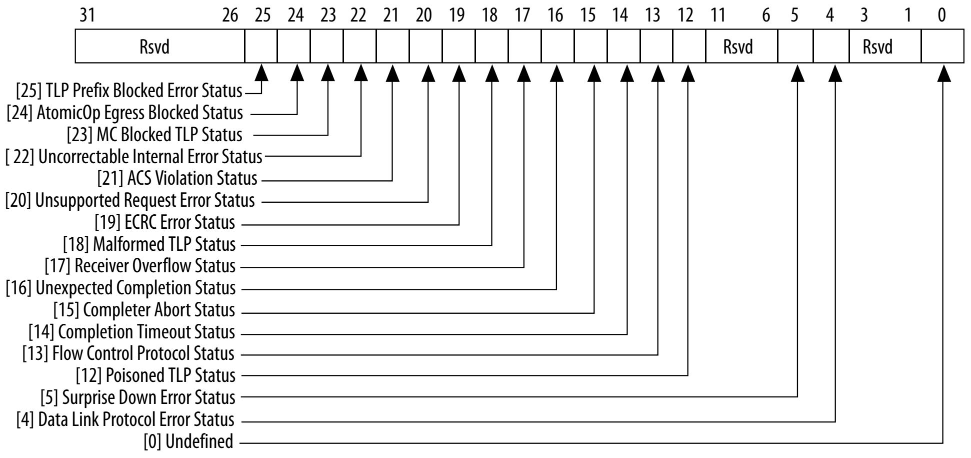
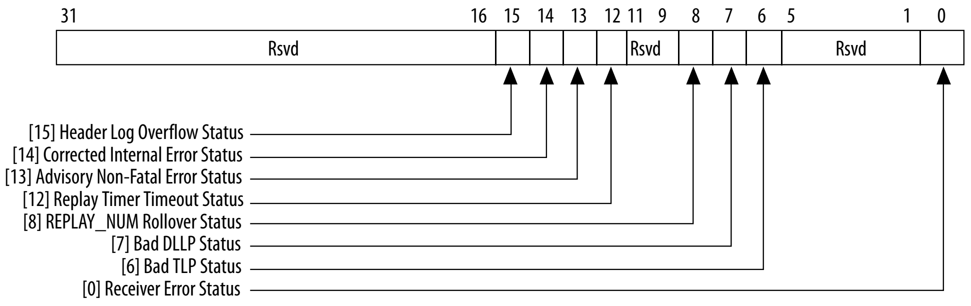

31
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
6
5
4
3
1
0
Rsvd
Rsvd
Rsvd
[25] TLP Prefix Blocked Error Status
[24] AtomicOp Egress Blocked Status
[23] MC Blocked TLP Status
[22] Uncorrectable Internal Error Status
[21] ACS Violation Status
[20] Unsupported Request Error Status
[19] ECRC Error Status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

Figure 54. 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.


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

Send Feedback

10. Design Implementation

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

10.1. Making Pin Assignments to Assign I/O Standard to Serial Data Pins

Before running Quartus Prime compilation, use the **Pin Planner** to assign I/O standards to the pins of the device.

1. On the Quartus Prime **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 the appropriate standard from the following table.

Table 74. I/O Standards for PCIe Pins

Pin Type	I/O Standard
PCIE REFCLK	**Current Mode Logic (CML)**, **HCSL**
PCIE RX	**Current Mode Logic (CML)** (8)
PCIE TX	**High Speed Differential I/O** (9)

The Quartus Prime software adds instance assignments to your Quartus Prime Settings File (`*.qsf`). The assignment is in the form `set_instance_assignment -name IO_STANDARD <"`*`IO_STANDARD_NAME`*`"> -to <`*`signal_name`*`>`. The `*.qsf` is in your synthesis directory.

Related Information

- Intel Arria 10 GX, GT, and SX Device Family Pin Connection Guidelines
  For information about connecting pins on the PCB including required resistor values and voltages.
- Intel Cyclone 10 GX Device Family Pin Connection Guidelines
  For information about connecting pins on the PCB including required resistor values and voltages.

(8) AC coupling is required at the transmitter for the PCIE RX signals.

(9) AC coupling is required at the transmitter for the PCIE TX signals.

10.2. Recommended Reset Sequence to Avoid Link Training Issues

Successful link training can only occur after the FPGA is configured. Designs using CvP for configuration initially load the I/O ring and periphery image. Intel Arria 10 or Intel Cyclone 10 GX devices include a Nios II Hard Calibration IP core that automatically calibrates the transceivers to optimize signal quality after CvP completes and before entering user mode. Link training occurs after calibration. 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.

Related Information

- Intel FPGA Intel Arria 10 Transceiver PHY IP Core User Guide
  For information about requirements for the CLKUSR pin used during automatic calibration.
- Intel FPGA Intel Cyclone 10 GX Transceiver PHY IP Core User Guide
  For information about requirements for the CLKUSR pin used during automatic calibration.

10.3. SDC Timing Constraints

Your top-level Synopsys Design Constraints file (`.sdc`) must include the following timing constraint macro for the Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCIe IP core.

Example 1. SDC Timing Constraints Required for the Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCIe and Design Example

```
# Constraints required for the 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
```

You should only include this constraint in one location across all of the SDC files in your project. Differences between Fitter timing analysis and Timing Analyzer timing analysis arise if these constraints are applied multiple times.

Related Information

What assignments do I need for a PCIe Gen1, Gen2 or Gen3 design that targets an Arria 10 ES2, ES3 or production device?
Starting with the Quartus Prime Software Release 17.0, these assignments are automatically included in the design. You do not have to add them.

11. Throughput Optimization

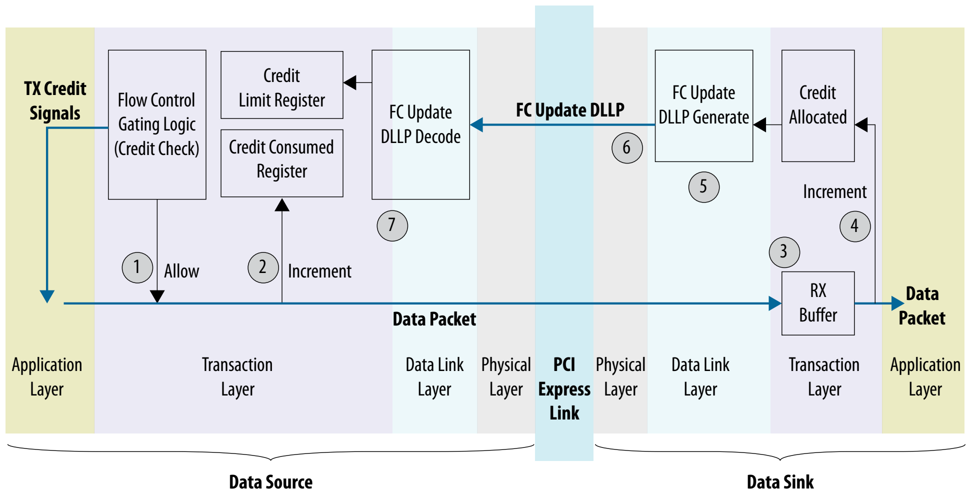
The *PCI Express Base Specification* defines a flow control mechanism to ensure efficient transfer of TLPs.

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 55. Flow Control Update Loop



The PCIe Hard IP maintains its own flow control logic, including a credit consumed register, and ensures that no TLP is sent that would use more credits than are available for that type of TLP. If you want optimum performance and granularity, you can maintain your own credit consumed register and flow control gating logic for each credit category (Header/Data, Posted/Non-posted/Completion). This allows you to halt the transmission of TLPs for a category that is out of credits, while still allowing TLP transmission for categories that have sufficient credits.

The following steps describe 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 required credits are less than or equal to the current value of available credits (credit limit - credits consumed so far), 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 original write requester receives the FC Update DLLP. 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.

11.1. Throughput of Posted Writes

The throughput of posted writes is limited primarily by the Flow Control Update loop as shown in Figure 55 on page 109. 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

11.2. 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 IP core 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.

12. Additional Features

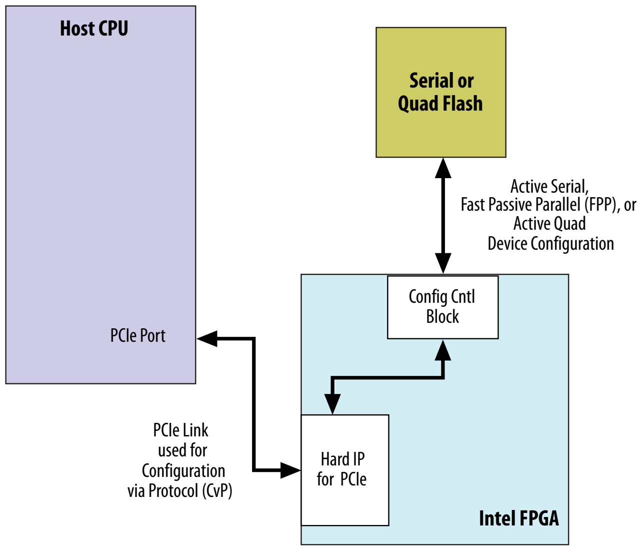
12.1. Configuration over Protocol (CvP)

The Hard IP for PCI Express architecture has an option to configure the FPGA and initialize 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. 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.

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 56. CvP in Intel Arria 10 or Intel Cyclone 10 GX Devices


Host CPU
PCIe Port
Serial or Quad Flash
Active Serial, Fast Passive Parallel (FPP), or Active Quad Device Configuration
Config Cntl Block
Hard IP for PCIe
Intel FPGA
PCIe Link used for Configuration via Protocol (CvP)

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**.
- May reduce system size because a single CvP link can be used to configure multiple FPGAs.

Note: The *Intel Cyclone 10 GX CvP Initialization over PCI Express User Guide* is now available.

Related Information

- Intel Arria 10 CvP Initialization and Partial Reconfiguration over PCI Express User Guide
- Intel Cyclone 10 GX CvP Initialization over PCI Express User Guide

12.2. Autonomous Mode

Autonomous mode allows the PCIe IP core to operate before the device enters user mode, while the core is being configured.

Intel's FPGA devices always receive the configuration bits for the periphery image first, then for the core fabric image. After the core image configures, the device enters user mode. In autonomous mode, the hard IP for PCI Express begins operation when the periphery configuration completes, before it enters user mode.

In autonomous mode, after completing link training, the Hard IP for PCI Express responds to Configuration Requests from the host with a Configuration Request Retry Status (CRRS). Autonomous mode is when you must meet the 100 ms PCIe wake-up time.

The hard IP for PCIe responds with CRRS under the following conditions:

- Before the core fabric is programmed when you enable autonomous mode.
- Before the core fabric is programmed when you enable initialization of the core fabric using the PCIe link.

All PCIe IP cores on a device can operate in autonomous mode. However, only the bottom Hard IP for PCI Express on either side can satisfy the 100 ms PCIe wake up time requirement. Transceiver calibration begins with the bottom PCIe IP core on each side of the device. Consequently, this IP core has a faster wake up time.

Note: If you enable the autonomous mode for the Hard IP for PCI Express, and also want to enable the Configuration Bypass feature, your design may not be able to link up to Gen 3 speed following a Cold Reset. In this case, a workaround is to configure the whole FPGA, then do a Warm Reset to allow the link to train to Gen 3 speed. Your design, including the autonomous Hard IP for PCI Express and Configuration Bypass, can then run at Gen 3 speed.

Arria V, Cyclone V, Stratix V, Intel Arria 10 and Intel Cyclone 10 GX devices are the first to offer autonomous mode. In earlier devices, the PCI Express Hard IP Core exits reset only after full FPGA configuration.

Related Information

- Enabling Autonomous Mode on page 115
- Enabling CvP Initialization on page 115

12.2.1. Enabling Autonomous Mode

These steps specify autonomous mode in the Quartus Prime software.

1. On the Quartus Prime Assignments menu, select **Device ➤ Device and Pin Options**.
2. Under **Category ➤ General** turn on **Enable autonomous PCIe HIP mode**. The **Enable autonomous PCIe HIP mode** option has an effect if your design has the following two characteristics:
    - You are using the Flash device or Ethernet controller, instead of the PCIe link to load the core image.
    - You have not turned on **Enable Configuration via the PCIe link** in the Hard IP for PCI Express GUI.

12.2.2. Enabling CvP Initialization

These steps enable CvP initialization mode in the Quartus Prime software.

1. On the Assignments menu select **Device ➤ Device and Pin Options.**
2. Under **Category**, select **CvP Settings**.
3. For **Configuration via Protocol**, select **Core initialization** from the drop-down menu.

12.3. 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**, and **ECRC generation** under the **PCI Express/PCI Capabilities** heading using the parameter editor to enable this functionality.

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

Related Information

PCI Express Base Specification 3.0

12.3.1. ECRC on the RX Path

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

Table 75. ECRC Operation on RX Path

ECRC Forwarding	ECRC Check Enable [10]	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

12.3.2. 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.

Table 76. 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 [11]	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	

continued...

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

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

ECRC Forwarding	ECRC Generation Enable [11]	TLP on Application	TLP on Link	Comments
	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	

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

13. Avalon-MM Testbench and Design Example

This chapter introduces the Endpoint design example including a testbench, BFM, and a test driver module. You can create this design example using design flows described in *Quick Start Guide*. This testbench uses the parameters that you specify in the *Quick Start Guide.*

When configured as an Endpoint variation, the testbench instantiates a design example and a Root Port BFM, which provides the following functions:

- A configuration routine that sets up all the basic configuration registers in the Endpoint. This configuration allows the Endpoint application to be the target and initiator of PCI Express transactions.
- A Verilog HDL procedure interface to initiate PCI Express transactions to the Endpoint.

The testbench uses a test driver module, **altera_avalon_dma** to exercise the DMA of the design example. The test driver module displays information from the Endpoint Configuration Space registers, so that you can correlate to the parameters you specify using the parameter editor.

- A configuration routine that sets up all the basic configuration registers in the Root Port and the Endpoint BFM. This configuration allows the Endpoint application to be the target and initiator of PCI Express transactions.
- A Verilog HDL procedure interface to initiate PCI Express transactions to the Endpoint BFM.

This testbench simulates a single Endpoint DUT.

The testbench uses a test driver module, `altpcietb_bfm_rp_gen3_x8.sv`, to exercise the target memory and DMA channel in the Endpoint BFM. The test driver module displays information from the Root Port Configuration Space registers, so that you can correlate to the parameters you specify using the parameter editor. The Endpoint model consists of an Endpoint variation combined with the DMA application.

Starting from the Intel Quartus Prime 18.0 release, you can generate an Intel Arria 10 PCIe example design that configures the IP as a Root Port. In this scenario, the testbench instantiates an Endpoint BFM and a JTAG master bridge.

The simulation uses the JTAG master BFM to initiate CRA read and write transactions to perform bus enumeration and configure the endpoint. The simulation also uses the JTAG master BFM to drive the TXS Avalon-MM interface to execute memory read and write transactions.

Note: The Intel testbench and Root Port BFM or Endpoint BFM provide a simple method to do basic testing of the Application Layer logic that interfaces to the variation. This BFM allows you to create and run simple task stimuli with configurable parameters to exercise basic functionality of the Intel example design. The testbench and BFM are not intended to be a substitute for a full verification environment. Corner cases and certain traffic profile stimuli are not covered. Refer to the items listed below for further details. To ensure the best verification coverage possible, Intel suggests strongly that you obtain commercially available PCI Express verification IP and tools, or do your own extensive hardware testing or both.

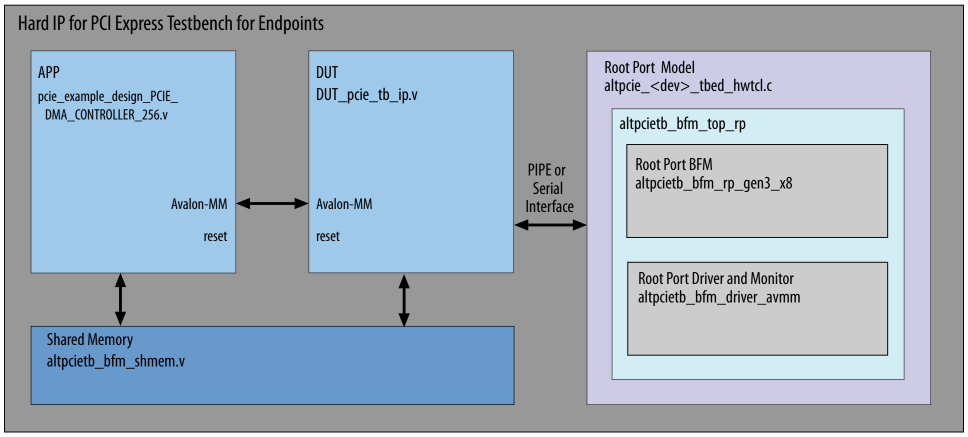
Your Application Layer design may need to handle at least the following scenarios that are not possible to create with the Intel testbench and the Root Port BFM:

- It is unable to generate or receive Vendor Defined Messages. Some systems generate Vendor Defined Messages and the Application Layer must be designed to process them. The Hard IP block passes these messages on to the Application Layer which, in most cases should ignore them.
- It can only handle received read requests that are less than or equal to the currently set equal to **Device ➤ PCI Express ➤ PCI Capabilities ➤ Maximum payload size** using the parameter editor. Many systems are capable of handling larger read requests that are then returned in multiple completions.
- It always returns a single completion for every read request. Some systems split completions on every 64-byte address boundary.
- It always returns completions in the same order the read requests were issued. Some systems generate the completions out-of-order.
- It is unable to generate zero-length read requests that some systems generate as flush requests following some write transactions. The Application Layer must be capable of generating the completions to the zero length read requests.
- It uses a fixed credit allocation.
- It does not support parity.
- It does not support multi-function designs which are available when using Configuration Space Bypass mode or Single Root I/O Virtualization (SR-IOV).

13.1. Avalon-MM Endpoint Testbench

You can generate the testbench from the example design by following the instructions in *Quick Start Guide*.

Figure 57. Design Example for Endpoint Designs



The Root Port BFM includes the following top-level modules in the `<`*`testbench_dir/`* `pcie_<`*`dev`*`>_hip_avmm_bridge_0_example_design/ pcie_example_design_tb/ip/pcie_example_design_tb/DUT_pcie_tb_ip/ altera_pcie_s10_tbed_<ver>/sim` directory:

- `altpcietb_bfm_top_rp.sv`: This is the Root Port PCI Express BFM. For more information about this module, refer to *Root Port BFM*.
- `altpcietb_bfm_rp_gen3_x8.sv`: This module drives transactions to the Root Port BFM. The main process operates in two stages:
  - First, it configures the Endpoint using the task `ebfm_cfg_rp_eg`.
  - Second, it runs a memory access test with the task `target_mem_test` or `target_mem_test_lite`.

    Finally, it runs a DMA test with the task `dma_mem_test`
    .

  This is the module that you modify to vary the transactions sent to the example Endpoint design or your own design.
- `altpcietb_bfm_shmem.v`: This memory implements the following functionality:
  - Provides data for TX write operations
  - Provides data for RX read operations
  - Receives data for RX write operations
  - Receives data for received completions

In addition, the testbench has routines that perform the following tasks:

- Generates the reference clock for the Endpoint at the required frequency.
- Provides a PCI Express reset at start up.

Note: Before running the testbench, you should set the `serial_sim_hwtcl` parameter in `<`*`testbench_dir`*`>/ pcie_<`*`dev`*`>_hip_avmm_bridge_example_design_tb/ip/ pcie_example_design_tb/DUT_pcie_tb_ip/ altera_pcie_<`*`dev`*`>_tbed_<`*`ver`*`>/sim/altpcie_<`*`dev`*`>_tbed_hwtcl.v`. Set to 1 for serial simulation and 0 for PIPE simulation.

13.2. Endpoint Design Example

This design example comprises a native Endpoint, a DMA application and a Root Port BFM. The write DMA module implements write operations from the Endpoint memory to the Root Complex (RC) memory. The read DMA implements read operations from the RC memory to the Endpoint memory.

When operating on a hardware platform, a software application running on the Root Complex processor typically controls the DMA. In simulation, the generated testbench, along with this design example, provide a BFM driver module in Verilog HDL that controls the DMA operations. Because the example relies on no other hardware interface than the PCI Express link, you can use the design example for the initial hardware validation of your system.

System generation creates the Endpoint variant in Verilog HDL. The testbench files are only available in Verilog HDL in the current release.

Note: The DMA design example requires setting BAR 2 or BAR 3 to a minimum of 256 bytes.

To run the DMA tests using MSI, you must set the **Number of MSI messages requested** parameter under the **PCI Express/PCI Capabilities** page to at least 2.

The DMA design example uses an architecture capable of transferring a large amount of fragmented memory without accessing the DMA registers for every memory block. For each memory block, the DMA design example uses a descriptor table containing the following information:

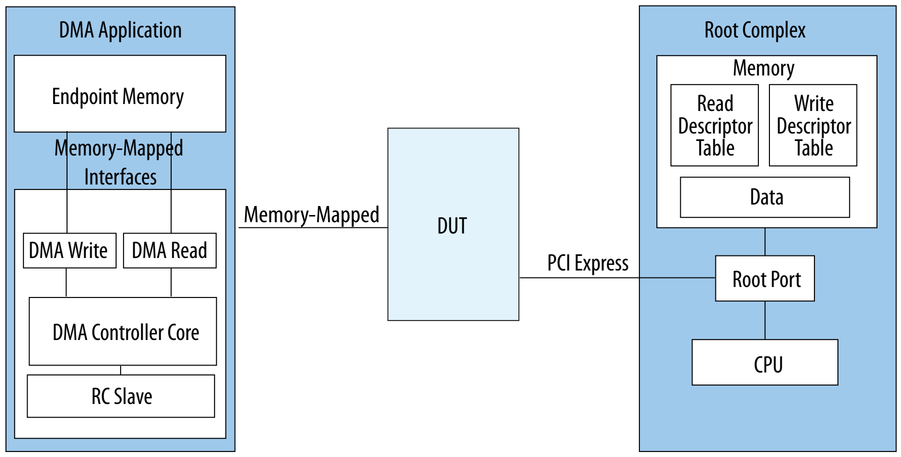
- Size of the transfer
- Address of the source
- Address of the destination
- Control bits to set the handshaking behavior between the software application or BFM driver and the DMA module

Note: The DMA design example only supports DWORD-aligned accesses. The DMA design example does not support ECRC forwarding.

The BFM driver writes the descriptor tables into BFM shared memory, from which the DMA design engine continuously collects the descriptor tables for DMA read, DMA write, or both. At the beginning of the transfer, the BFM programs the Endpoint DMA control register. The DMA control register indicates the total number of descriptor tables and the BFM shared memory address of the first descriptor table. After programming the DMA control register, the DMA engine continuously fetches descriptors from the BFM shared memory for both DMA reads and DMA writes, and then performs the data transfer for each descriptor.

The following figure shows a block diagram of the design example connected to an external RC CPU.

Figure 58. Top-Level DMA Example for Simulation



The block diagram contains the following elements:

- The DMA application connects to the Avalon-MM interface of the Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express. The connections consist of the following interfaces:
  - The Avalon-MM RX master receives TLP header and data information from the Hard IP block.
  - The Avalon-MM TX slave transmits TLP header and data information to the Hard IP block.
  - The Avalon-MM control register access (CRA) IRQ port requests MSI interrupts from the Hard IP block.
  - The sideband signal bus carries static information such as configuration information.
- The BFM shared memory stores the descriptor tables for the DMA read and the DMA write operations.
- A Root Complex CPU and associated PCI Express PHY connect to the Endpoint design example, using a Root Port.

The example Endpoint design and application accomplish the following objectives:

- Show you how to interface to the Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express using the Avalon-MM protocol.
- Provide a DMA channel that initiates memory read and write transactions on the PCI Express link.

The DMA design example hierarchy consists of these components:

- A DMA read and a DMA write module
- An on-chip Endpoint memory (Avalon-MM slave) which uses two Avalon-MM interfaces for each engine

The RC slave module typically drives downstream transactions which target the Endpoint on-chip buffer memory. These target memory transactions bypass the DMA engines. In addition, the RC slave module monitors performance and acknowledges incoming message TLPs.

Related Information

Embedded Peripherals IP User Guide Introduction
For more information about the DMA Controller.

13.2.1. BAR/Address Map

The design example maps received memory transactions to either the target memory block or the control register block based on which BAR the transaction matches. There are multiple BARs that map to each of these blocks to maximize interoperability with different variation files. The following table shows the mapping.

Table 77. BAR Map

Memory BAR	Mapping
32-bit BAR0 32-bit BAR1 64-bit BAR1:0	Maps to 32 KB target memory block. Use the rc_slave module to bypass the chaining DMA.
32-bit BAR2 32-bit BAR3 64-bit BAR3:2	Maps to DMA Read and DMA write control and status registers, a minimum of 256 bytes.
32-bit BAR4 32-bit BAR5 64-bit BAR5:4	Maps to 32 KB target memory block. Use the rc_slave module to bypass the chaining DMA.
Expansion ROM BAR	Not implemented by design example; behavior is unpredictable.
I/O Space BAR (any)	Not implemented by design example; behavior is unpredictable.

13.2.2. BAR Setup

The `find_mem_bar` task in Root Port BFM `altpcietb_bfm_rp_gen3_x8.sv` sets up BARs to match your design.

13.3. Avalon-MM Test Driver Module

The BFM driver module, **altpcietb_bfm_driver_avmm.v** is configured to test the DMA example Endpoint design. The BFM driver module configures the Endpoint Configuration Space registers and then tests the example Endpoint DMA channel. This file is in the `<`*`variation_name`*`>_tb/altera_pcie_<`*`dev`*`>_tbed_<`*`quartus_ver`*`>/sim/` directory.

The BFM test driver module performs the following steps in sequence:

1. Configures the Root Port and Endpoint Configuration Spaces, which the BFM test driver module does by calling the procedure `ebfm_cfg_rp_ep`, which is part of **altpcietb_bfm_configure**.
2. Finds a suitable BAR to access the example Endpoint design Control Register space. Either BARs 2 or 3 must be at least a 256-byte memory BAR to perform the DMA channel test. The `find_mem_bar` procedure in the **altpcietb_bfm_driver_avmm** does this.
3. If a suitable BAR is found in the previous step, the driver performs the following tasks:

a. DMA read—The driver programs the DMA to read data from the BFM shared memory into the Endpoint memory. The descriptor control fields specify for the DMA to issue an MSI when the last descriptor has completed.

a. DMA write—The driver programs the DMA to write the data from its Endpoint memory back to the BFM shared memory. The descriptor control fields are specified so that the DMA completes the following steps to indicate transfer completion:

   - The DMA issues an MSI when the last descriptor has completed.
   - The data written back to BFM is checked against the data that was read from the BFM.
   - The driver programs the DMA to perform a test that demonstrates downstream access of the DMA Endpoint memory.

Note: Edit this file if you want to add your own custom PCIe transactions. Insert your own custom function after the `find_mem_bar` function. You can use the functions in the *BFM Procedures and Functions* section.

Related Information

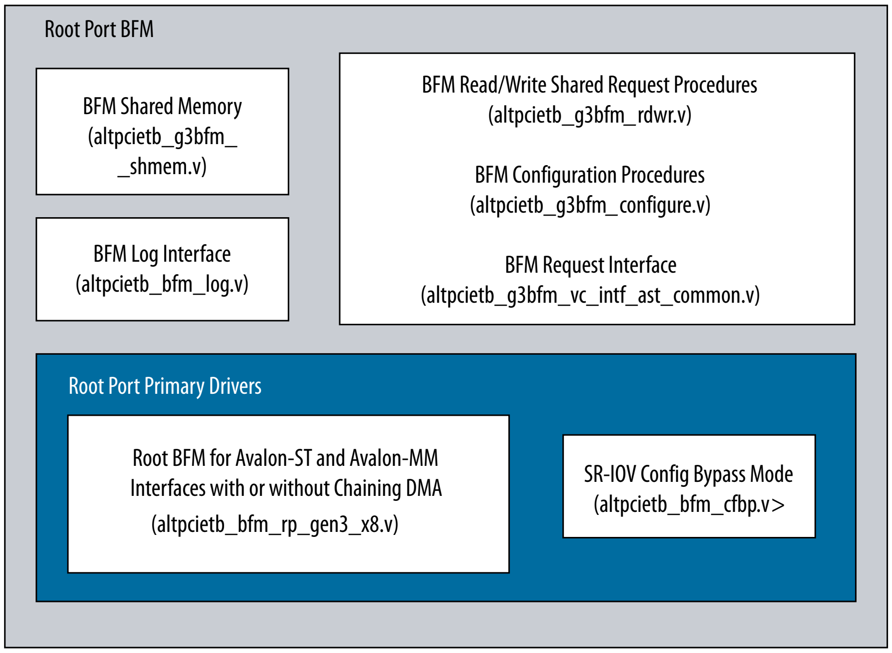
BFM Procedures and Functions on page 133

13.4. Root Port BFM

13.4.1. Root Port BFM Overview

The basic Root Port BFM provides Verilog HDL task-based interface to request transactions to issue on the PCI Express link. The Root Port BFM also handles requests received from the PCI Express link. The following figure shows the most important modules in the Root Port BFM.

Figure 59. Root Port BFM


Root Port BFM
BFM Shared Memory (altpcietb_g3bfm_ _shmem.v)
BFM Log Interface (altpcietb_bfm_log.v)
BFM Read/Write Shared Request Procedures (altpcietb_g3bfm_rdwr.v)
BFM Configuration Procedures (altpcietb_g3bfm_configure.v)
BFM Request Interface (altpcietb_g3bfm_vc_intf_ast_common.v)
Root Port Primary Drivers
Root BFM for Avalon-ST and Avalon-MM Interfaces with or without Chaining DMA (altpcietb_bfm_rp_gen3_x8.v)
SR-IOV Config Bypass Mode (altpcietb_bfm_cfbp.v>

These modules implement the following functionality:

- BFM Log Interface,`altpcietb_bfm_log.v` and `altlpcietb_bfm_rp_<`*gen*`>_x8.v`: The BFM log functions provides routine for writing commonly formatted messages to the simulator standard output and optionally to a log file. It also provides controls that stop simulation on errors. For details on these procedures, refer to *BFM Log and Message Procedures*.
- BFM Read/Write Request Functions, `altpcietb_bfm_rp_<`*gen*`>_x8.sv`: These functions provide the basic BFM calls for PCI Express read and write requests. For details on these procedures, refer to *BFM Read and Write Procedures*.
- BFM Log Interface, `altpcietb_bfm_log.v` and `altlpcietb_bfm_rp_<`*gen*`>_x8.v`: The BFM log functions provides routine for writing commonly formatted messages to the simulator standard output and optionally to a log file. It also provides controls that stop simulation on errors. For details on these procedures, refer to *BFM Log and Message Procedures*.
- BFM Configuration Functions, `altpcietb_g3bfm_configure.v` : These functions provide the BFM calls to request configuration of the PCI Express link and the Endpoint Configuration Space registers. For details on these procedures and functions, refer to *BFM Configuration Procedures*.

- BFM shared memory, `altpcietb_g3bfm_shmem_common.v`: This modules provides the Root Port BFM shared memory. It implements the following functionality:
  - Provides data for TX write operations
  - Provides data for RX read operations
  - Receives data for RX write operations
  - Receives data for received completions

  Refer to *BFM Shared Memory Access Procedures* to learn more about the procedures to read, write, fill, and check the shared memory from the BFM driver.
- BFM Request Interface, `altpcietb_g3bfm_req_intf.v`: This interface provides the low-level interface between the `altpcietb_g3bfm_rdwr` and `altpcietb_bfm_configure` procedures or functions and the Root Port RTL Model. This interface stores a write-protected data structure containing the sizes and the values programmed in the BAR registers of the Endpoint. It also stores other critical data used for internal BFM management. You do not need to access these files directly to adapt the testbench to test your Endpoint application.
- Avalon-ST Interfaces, `altpcietb_g3bfm_vc_intf_ast_common.v`: These interface modules handle the Root Port interface model. They take requests from the BFM request interface and generate the required PCI Express transactions. They handle completions received from the PCI Express link and notify the BFM request interface when requests are complete. Additionally, they handle any requests received from the PCI Express link, and store or fetch data from the shared memory before generating the required completions.

Related Information

- Test Signals on page 63
- BFM Shared Memory Access Procedures on page 138

13.4.2. Issuing Read and Write Transactions to the Application Layer

The Endpoint Application Layer issues read and write transactions by calling one of the `ebfm_bar` procedures in `altpcietb_g3bfm_rdwr.v`. The procedures and functions listed below are available in the Verilog HDL include file `altpcietb_g3bfm_rdwr.v`. The complete list of available procedures and functions is as follows:

- `ebfm_barwr`: writes data from BFM shared memory to an offset from a specific Endpoint BAR. This procedure returns as soon as the request has been passed to the VC interface module for transmission.
- `ebfm_barwr_imm`: writes a maximum of four bytes of immediate data (passed in a procedure call) to an offset from a specific Endpoint BAR. This procedure returns as soon as the request has been passed to the VC interface module for transmission.
- `ebfm_barrd_wait`: reads data from an offset of a specific Endpoint BAR and stores it in BFM shared memory. This procedure blocks waiting for the completion data to be returned before returning control to the caller.
- `ebfm_barrd_nowt`: reads data from an offset of a specific Endpoint BAR and stores it in the BFM shared memory. This procedure returns as soon as the request has been passed to the VC interface module for transmission, allowing subsequent reads to be issued in the interim.

These routines take as parameters a BAR number to access the memory space and the BFM shared memory address of the `bar_table` data structure that was set up by the `ebfm_cfg_rp_ep` procedure. (Refer to *Configuration of Root Port and Endpoint*.) Using these parameters simplifies the BFM test driver routines that access an offset from a specific BAR and eliminates calculating the addresses assigned to the specified BAR.

The Root Port BFM does not support accesses to Endpoint I/O space BARs.

Related Information

Configuration of Root Port and Endpoint on page 127

13.4.3. Configuration of Root Port and Endpoint

Before you issue transactions to the Endpoint, you must configure the Root Port and Endpoint Configuration Space registers. Use `ebfm_cfg_rp_ep` in `altpcietb_bfm_configure.v` to configure these registers.

The `ebfm_cfg_rp_ep` procedure executes the following steps to initialize the Configuration Space:

1. Sets the Root Port Configuration Space to enable the Root Port to send transactions on the PCI Express link.
2. Sets the Root Port and Endpoint PCI Express Capability Device Control registers as follows:
    a. Disables `Error Reporting` in both the Root Port and Endpoint. The BFM does not have error handling capability.
    b. Enables `Relaxed Ordering` in both Root Port and Endpoint.
    c. Enables `Extended Tags` for the Endpoint if the Endpoint has that capability.
    d. Disables `Phantom Functions`, `Aux Power PM`, and `No Snoop` in both the Root Port and Endpoint.
    e. Sets the `Max Payload Size` to the value that the Endpoint supports because the Root Port supports the maximum payload size.
    f. Sets the Root Port `Max Read Request Size` to 4 KB because the example Endpoint design supports breaking the read into as many completions as necessary.
    g. Sets the Endpoint `Max Read Request Size` equal to the `Max Payload Size` because the Root Port does not support breaking the read request into multiple completions.
3. Assigns values to all the Endpoint BAR registers. The BAR addresses are assigned by the algorithm outlined below.

   a. I/O BARs are assigned smallest to largest starting just above the ending address of BFM shared memory in I/O space and continuing as needed throughout a full 32-bit I/O space.
   b. The 32-bit non-prefetchable memory BARs are assigned smallest to largest, starting just above the ending address of BFM shared memory in memory space and continuing as needed throughout a full 32-bit memory space.
   c. The value of the `addr_map_4GB_limit` input to the `ebfm_cfg_rp_ep` procedure controls the assignment of the 32-bit prefetchable and 64-bit prefetchable memory BARS. The default value of the `addr_map_4GB_limit` is `0`.

      If the `addr_map_4GB_limit` input to the `ebfm_cfg_rp_ep` procedure is set to 0, then the `ebfm_cfg_rp_ep` procedure assigns the 32-bit prefetchable memory BARs largest to smallest, starting at the top of 32-bit memory space and continuing as needed down to the ending address of the last 32-bit non-prefetchable BAR.

      However, if the `addr_map_4GB_limit` input is set to 1, the address map is limited to 4 GB. The `ebfm_cfg_rp_ep` procedure assigns 32-bit and 64-bit prefetchable memory BARs largest to smallest, starting at the top of the 32-bit memory space and continuing as needed down to the ending address of the last 32-bit non-prefetchable BAR.
   d. If the `addr_map_4GB_limit` input to the `ebfm_cfg_rp_ep` procedure is set to 0, then the `ebfm_cfg_rp_ep` procedure assigns the 64-bit prefetchable memory BARs smallest to largest starting at the 4 GB address assigning memory ascending above the 4 GB limit throughout the full 64-bit memory space.

      If the `addr_map_4 GB_limit` input to the `ebfm_cfg_rp_ep` procedure is set to 1, the `ebfm_cfg_rp_ep` procedure assigns the 32-bit and the 64-bit prefetchable memory BARs largest to smallest starting at the 4 GB address and assigning memory by descending below the 4 GB address to memory addresses as needed down to the ending address of the last 32-bit non-prefetchable BAR.

      The above algorithm cannot always assign values to all BARs when there are a few very large (1 GB or greater) 32-bit BARs. Although assigning addresses to all BARs may be possible, a more complex algorithm would be required to effectively assign these addresses. However, such a configuration is unlikely to be useful in real systems. If the procedure is unable to assign the BARs, it displays an error message and stops the simulation.

4. Based on the above BAR assignments, the `ebfm_cfg_rp_ep` procedure assigns the Root Port Configuration Space address windows to encompass the valid BAR address ranges.
5. The `ebfm_cfg_rp_ep` procedure enables master transactions, memory address decoding, and I/O address decoding in the Endpoint PCIe control register.

The `ebfm_cfg_rp_ep` procedure also sets up a `bar_table` data structure in BFM shared memory that lists the sizes and assigned addresses of all Endpoint BARs. This area of BFM shared memory is write-protected. Consequently, any application logic write accesses to this area cause a fatal simulation error.

BFM procedure calls to generate full PCIe addresses for read and write requests to particular offsets from a BAR use this data structure. . This procedure allows the testbench code that accesses the Endpoint application logic to use offsets from a BAR and avoid tracking specific addresses assigned to the BAR. The following table shows how to use those offsets.

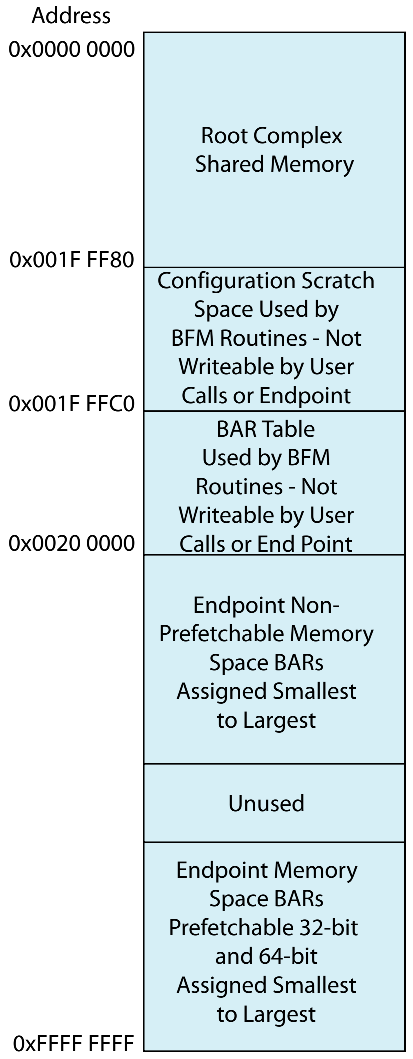
Table 78. BAR Table Structure

Offset (Bytes)	Description
+0	PCI Express address in BAR0
+4	PCI Express address in BAR1
+8	PCI Express address in BAR2
+12	PCI Express address in BAR3
+16	PCI Express address in BAR4
+20	PCI Express address in BAR5
+24	PCI Express address in Expansion ROM BAR
+28	Reserved
+32	BAR0 read back value after being written with all 1's (used to compute size)
+36	BAR1 read back value after being written with all 1's
+40	BAR2 read back value after being written with all 1's
+44	BAR3 read back value after being written with all 1's
+48	BAR4 read back value after being written with all 1's
+52	BAR5 read back value after being written with all 1's
+56	Expansion ROM BAR read back value after being written with all 1's
+60	Reserved

The configuration routine does not configure any advanced PCI Express capabilities such as the AER capability.

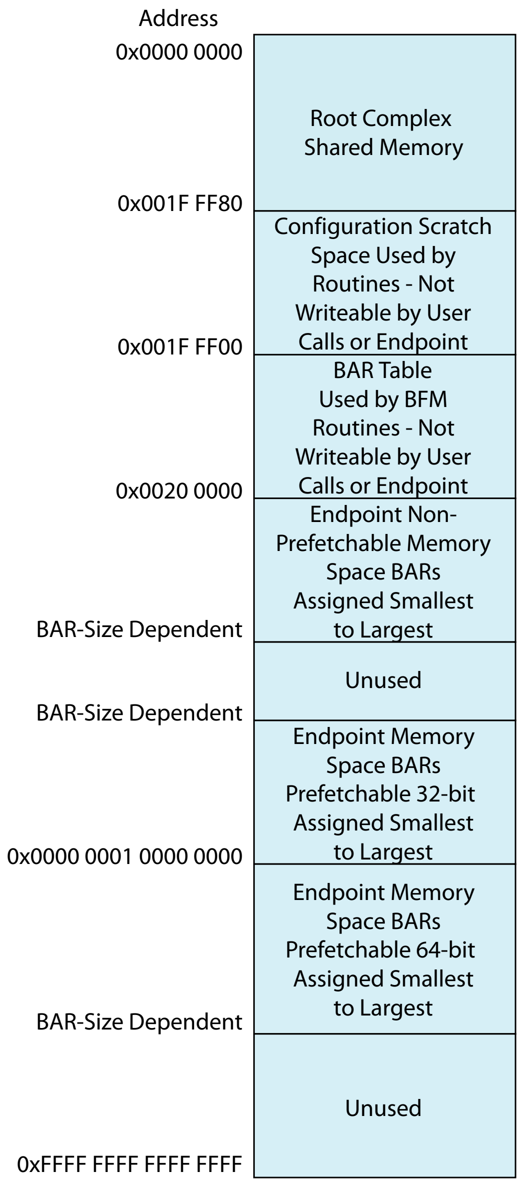
Besides the `ebfm_cfg_rp_ep` procedure in `altpcietb_bfm_rp_gen3_x8.sv`, routines to read and write Endpoint Configuration Space registers directly are available in the Verilog HDL include file. After the `ebfm_cfg_rp_ep` procedure runs the PCI Express I/O and Memory Spaces have the layout shown in the following three figures. The memory space layout depends on the value of the **addr_map_4GB_limit** input parameter. The following figure shows the resulting memory space map when the **addr_map_4GB_limit** is 1.

Figure 60. Memory Space Layout—4 GB Limit


Address
0x0000 0000
Root Complex Shared Memory
0x001F FF80
Configuration Scratch Space Used by BFM Routines - Not Writeable by User Calls or Endpoint
0x001F FFC0
BAR Table Used by BFM Routines - Not Writeable by User Calls or End Point
0x0020 0000
Endpoint Non-Prefetchable Memory Space BARs Assigned Smallest to Largest
Unused
Endpoint Memory Space BARs Prefetchable 32-bit and 64-bit Assigned Smallest to Largest
0xFFFF FFFF

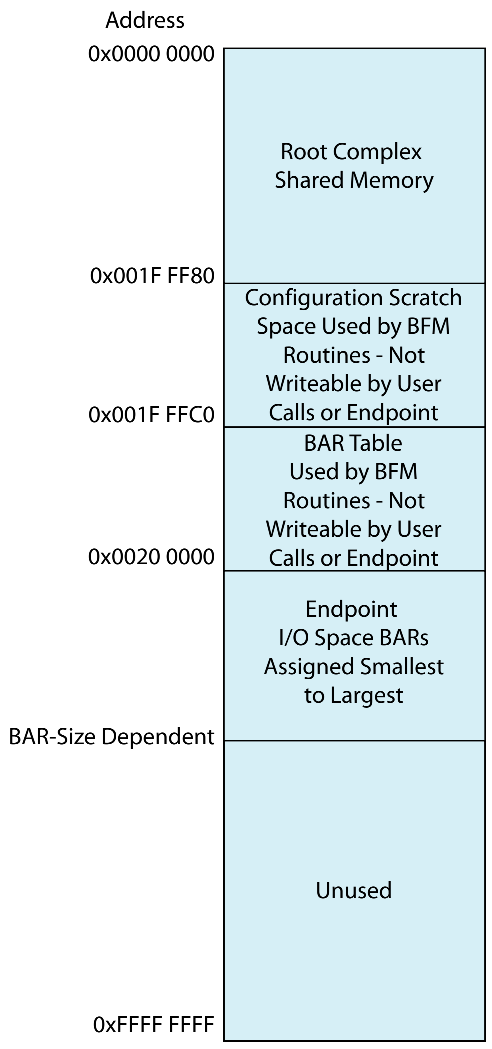
The following figure shows the resulting memory space map when the **addr_map_4GB_limit** is 0.

Figure 61. Memory Space Layout—No Limit


Address
0x0000 0000
Root Complex Shared Memory
0x001F FF80
Configuration Scratch Space Used by Routines - Not Writeable by User Calls or Endpoint
0x001F FF00
BAR Table Used by BFM Routines - Not Writeable by User Calls or Endpoint
0x0020 0000
Endpoint Non-Prefetchable Memory Space BARs Assigned Smallest to Largest
BAR-Size Dependent
Unused
BAR-Size Dependent
Endpoint Memory Space BARs Prefetchable 32-bit Assigned Smallest to Largest
0x0000 0001 0000 0000
Endpoint Memory Space BARs Prefetchable 64-bit Assigned Smallest to Largest
BAR-Size Dependent
Unused
0xFFFF FFFF FFFF FFFF

The following figure shows the I/O address space.

Figure 62. I/O Address Space


Address
0x0000 0000
Root Complex Shared Memory
0x001F FF80
Configuration Scratch Space Used by BFM Routines - Not Writeable by User Calls or Endpoint
0x001F FFC0
BAR Table Used by BFM Routines - Not Writeable by User Calls or Endpoint
0x0020 0000
Endpoint I/O Space BARs Assigned Smallest to Largest
BAR-Size Dependent
Unused
0xFFFF FFFF

13.4.4. Configuration Space Bus and Device Numbering

Enumeration assigns the Root Port interface device number 0 on internal bus number 0. Use the `ebfm_cfg_rp_ep` to assign the Endpoint to any device number on any bus number (greater than 0). The specified bus number is the secondary bus in the Root Port Configuration Space.

13.4.5. BFM Memory Map

The BFM shared memory is 2 MBs. The BFM shared memory maps to the first 2 MBs of I/O space and also the first 2 MBs of memory space. When the Endpoint application generates an I/O or memory transaction in this range, the BFM reads or writes the shared memory.

13.5. BFM Procedures and Functions

The BFM includes procedures, functions, and tasks to drive Endpoint application testing. It also includes procedures to run the chaining DMA design example.

The BFM read and write procedures read and write data to BFM shared memory, Endpoint BARs, and specified configuration registers. The procedures and functions are available in the Verilog HDL. These procedures and functions support issuing memory and configuration transactions on the PCI Express link.

13.5.1. ebfm_barwr Procedure

The `ebfm_barwr` procedure writes a block of data from BFM shared memory to an offset from the specified Endpoint BAR. The length can be longer than the configured `MAXIMUM_PAYLOAD_SIZE`. The procedure breaks the request up into multiple transactions as needed. This routine returns as soon as the last transaction has been accepted by the VC interface module.

Location	altpcietb_bfm_rdwr.v	
Syntax	`ebfm_barwr(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory. The `bar_table` structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	`bar_num`	Number of the BAR used with `pcie_offset` to determine PCI Express address.
	`pcie_offset`	Address offset from the BAR base.
	`lcladdr`	BFM shared memory address of the data to be written.
	`byte_len`	Length, in bytes, of the data written. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	`tclass`	Traffic class used for the PCI Express transaction.

13.5.2. ebfm_barwr_imm Procedure

The `ebfm_barwr_imm` procedure writes up to four bytes of data to an offset from the specified Endpoint BAR.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`ebfm_barwr_imm(bar_table, bar_num, pcie_offset, imm_data, byte_len, tclass)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory. The `bar_table` structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	`bar_num`	Number of the BAR used with `pcie_offset` to determine PCI Express address.
	`pcie_offset`	Address offset from the BAR base.

continued...

Location	altpcietb_bfm_driver_rp.v	
	imm_data	Data to be written. In Verilog HDL, this argument is reg [31:0].In both languages, the bits written depend on the length as follows: Length Bits Written • 4: 31 down to 0 • 3: 23 down to 0 • 2: 15 down to 0 • 1: 7 down to 0
	byte_len	Length of the data to be written in bytes. Maximum length is 4 bytes.
	tclass	Traffic class to be used for the PCI Express transaction.

13.5.3. ebfm_barrd_wait Procedure

The ebfm_barrd_wait procedure reads a block of data from the offset of the specified Endpoint BAR and stores it in BFM shared memory. The length can be longer than the configured maximum read request size; the procedure breaks the request up into multiple transactions as needed. This procedure waits until all of the completion data is returned and places it in shared memory.

Location	altpcietb_bfm_driver_rp.v altpcietb_bfm_rdwr.v	
Syntax	ebfm_barrd_wait(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory. The bar_table structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.
	pcie_offset	Address offset from the BAR base.
	lcladdr	BFM shared memory address where the read data is stored.
	byte_len	Length, in bytes, of the data to be read. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	tclass	Traffic class used for the PCI Express transaction.

13.5.4. ebfm_barrd_nowt Procedure

The ebfm_barrd_nowt procedure reads a block of data from the offset of the specified Endpoint BAR and stores the data in BFM shared memory. The length can be longer than the configured maximum read request size; the procedure breaks the request up into multiple transactions as needed. This routine returns as soon as the last read transaction has been accepted by the VC interface module, allowing subsequent reads to be issued immediately.

Location	altpcietb_bfm_driver_rp.v altpcietb_bfm_rdwr.v	
Syntax	ebfm_barrd_nowt(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory.
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.

continued...

Send Feedback

Location	**altpcietb_bfm_driver_rp.v** **altpcietb_bfm_rdwr.v**	
	`pcie_offset`	Address offset from the BAR base.
	`lcladdr`	BFM shared memory address where the read data is stored.
	`byte_len`	Length, in bytes, of the data to be read. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	`tclass`	Traffic Class to be used for the PCI Express transaction.

13.5.5. ebfm_cfgwr_imm_wait Procedure

The `ebfm_cfgwr_imm_wait` procedure writes up to four bytes of data to the specified configuration register. This procedure waits until the write completion has been returned.

Location	**altpcietb_bfm_driver_rp.v** **altpcietb_bfm_rdwr.v**	
Syntax	`ebfm_cfgwr_imm_wait(bus_num, dev_num, fnc_num, imm_regb_ad, regb_ln, imm_data, compl_status`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data written. Maximum length is four bytes. The `regb_ln` and the `regb_ad` arguments cannot cross a DWORD boundary.
	`imm_data`	Data to be written. This argument is `reg [31:0]`. The bits written depend on the length: • 4: 31 down to 0 • 3: 23 down to 0 • 2: 15 down to 0 • 1: 7 down to 0
	`compl_status`	This argument is `reg [2:0]`. This argument is the completion status as specified in the PCI Express specification. The following encodings are defined: • 3’b000: SC— Successful completion • 3’b001: UR— Unsupported Request • 3’b010: CRS — Configuration Request Retry Status • 3’b100: CA — Completer Abort

13.5.6. ebfm_cfgwr_imm_nowt Procedure

The `ebfm_cfgwr_imm_nowt` procedure writes up to four bytes of data to the specified configuration register. This procedure returns as soon as the VC interface module accepts the transaction, allowing other writes to be issued in the interim. Use this procedure only when successful completion status is expected.

Location	altpcietb_bfm_driver_rp.v altpcietb_bfm_rdwr.v	
Syntax	`ebfm_cfgwr_imm_nowt(bus_num, dev_num, fnc_num, imm_regb_adr, regb_len, imm_data)`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data written. Maximum length is four bytes, The `regb_ln` the `regb_ad` arguments cannot cross a DWORD boundary.
	`imm_data`	Data to be written This argument is `reg [31:0]`. In both languages, the bits written depend on the length. The following encodes are defined. • 4: [31:0] • 3: [23:0] • 2: [15:0] • 1: [7:0]

13.5.7. ebfm_cfgrd_wait Procedure

The `ebfm_cfgrd_wait` procedure reads up to four bytes of data from the specified configuration register and stores the data in BFM shared memory. This procedure waits until the read completion has been returned.

Location	altpcietb_bfm_driver_rp.v altpcietb_bfm_rdwr.v	
Syntax	`ebfm_cfgrd_wait(bus_num, dev_num, fnc_num, regb_ad, regb_ln, lcladdr, compl_status)`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data read. Maximum length is four bytes. The `regb_ln` and the `regb_ad` arguments cannot cross a DWORD boundary.
	`lcladdr`	BFM shared memory address of where the read data should be placed.
	`compl_status`	Completion status for the configuration transaction. This argument is reg [2:0]. In both languages, this is the completion status as specified in the PCI Express specification. The following encodings are defined. • 3'b000: SC— Successful completion • 3'b001: UR— Unsupported Request • 3'b010: CRS — Configuration Request Retry Status • 3'b100: CA — Completer Abort

13.5.8. ebfm_cfgrd_nowt Procedure

The `ebfm_cfgrd_nowt` procedure reads up to four bytes of data from the specified configuration register and stores the data in the BFM shared memory. This procedure returns as soon as the VC interface module has accepted the transaction, allowing other reads to be issued in the interim. Use this procedure only when successful completion status is expected and a subsequent read or write with a wait can be used to guarantee the completion of this operation.

Location	altpcietb_bfm_driver_rp.v altpcietb_bfm_rdwr.v	
Syntax	`ebfm_cfgrd_nowt(bus_num, dev_num, fnc_num, regb_ad, regb_ln, lcladdr)`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data written. Maximum length is four bytes. The `regb_ln` and `regb_ad` arguments cannot cross a DWORD boundary.
	`lcladdr`	BFM shared memory address where the read data should be placed.

13.5.9. BFM Configuration Procedures

The BFM configuration procedures are available in `altpcietb_bfm_configure.v`. These procedures support configuration of the Root Port and Endpoint Configuration Space registers.

All Verilog HDL arguments are type `integer` and are input-only unless specified otherwise.

13.5.9.1. ebfm_cfg_rp_ep Procedure

The `ebfm_cfg_rp_ep` procedure configures the Root Port and Endpoint Configuration Space registers for operation.

Location	altpcietb_bfm_configure.v	
Syntax	`ebfm_cfg_rp_ep(bar_table, ep_bus_num, ep_dev_num, rp_max_rd_req_size, display_ep_config, addr_map_4GB_limit)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory. This routine populates the `bar_table` structure. The `bar_table` structure stores the size of each BAR and the address values assigned to each BAR. The address of the `bar_table` structure is passed to all subsequent read and write procedure calls that access an offset from a particular BAR.
	`ep_bus_num`	PCI Express bus number of the target device. This number can be any value greater than 0. The Root Port uses this as the secondary bus number.
	`ep_dev_num`	PCI Express device number of the target device. This number can be any value. The Endpoint is automatically assigned this value when it receives the first configuration transaction.

continued...

Location	altpcietb_bfm_configure.v	
	rp_max_rd_req_size	Maximum read request size in bytes for reads issued by the Root Port. This parameter must be set to the maximum value supported by the Endpoint Application Layer. If the Application Layer only supports reads of the MAXIMUM_PAYLOAD_SIZE, then this can be set to 0 and the read request size is set to the maximum payload size. Valid values for this argument are 0, 128, 256, 512, 1,024, 2,048 and 4,096.
	display_ep_config	When set to 1 many of the Endpoint Configuration Space registers are displayed after they have been initialized, causing some additional reads of registers that are not normally accessed during the configuration process such as the Device ID and Vendor ID.
	addr_map_4GB_limit	When set to 1 the address map of the simulation system is limited to 4 GB. Any 64-bit BARs are assigned below the 4 GB limit.

13.5.9.2. ebfm_cfg_decode_bar Procedure

The `ebfm_cfg_decode_bar` procedure analyzes the information in the BAR table for the specified BAR and returns details about the BAR attributes.

Location	altpcietb_bfm_configure.v	
Syntax	ebfm_cfg_decode_bar(bar_table, bar_num, log2_size, is_mem, is_pref, is_64b)	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory.
	bar_num	BAR number to analyze.
	log2_size	This argument is set by the procedure to the log base 2 of the size of the BAR. If the BAR is not enabled, this argument is set to 0.
	is_mem	The procedure sets this argument to indicate if the BAR is a memory space BAR (1) or I/O Space BAR (0).
	is_pref	The procedure sets this argument to indicate if the BAR is a prefetchable BAR (1) or non-prefetchable BAR (0).
	is_64b	The procedure sets this argument to indicate if the BAR is a 64-bit BAR (1) or 32-bit BAR (0). This is set to 1 only for the lower numbered BAR of the pair.

13.5.10. BFM Shared Memory Access Procedures

These procedures and functions support accessing the BFM shared memory.

13.5.10.1. Shared Memory Constants

The following constants are defined in `altpcietb_bfm_driver.v`. They select a data pattern in the `shmem_fill` and `shmem_chk_ok` routines. These shared memory constants are all Verilog HDL type `integer`.

Table 79. Constants: Verilog HDL Type INTEGER

Constant	Description
SHMEM_FILL_ZEROS	Specifies a data pattern of all zeros
SHMEM_FILL_BYTE_INC	Specifies a data pattern of incrementing 8-bit bytes (0x00, 0x01, 0x02, etc.)
SHMEM_FILL_WORD_INC	Specifies a data pattern of incrementing 16-bit words (0x0000, 0x0001, 0x0002, etc.)

continued...

Constant	Description
SHMEM_FILL_DWORD_INC	Specifies a data pattern of incrementing 32-bit DWORDs (0x00000000, 0x00000001, 0x00000002, etc.)
SHMEM_FILL_QWORD_INC	Specifies a data pattern of incrementing 64-bit qwords (0x0000000000000000, 0x0000000000000001, 0x0000000000000002, etc.)
SHMEM_FILL_ONE	Specifies a data pattern of all ones

13.5.10.2. shmem_write Task

The `shmem_write` procedure writes data to the BFM shared memory.

Location	altpcietb_bfm_shmem.v	
Syntax	`shmem_write(addr, data, leng)`	
Arguments	`addr`	BFM shared memory starting address for writing data
	`data`	Data to write to BFM shared memory. This parameter is implemented as a 64-bit vector. `leng` is 1–8 bytes. Bits 7 down to 0 are written to the location specified by `addr`; bits 15 down to 8 are written to the `addr+1` location, etc.
	`length`	Length, in bytes, of data written

13.5.10.3. shmem_read Function

The `shmem_read` function reads data to the BFM shared memory.

Location	altpcietb_bfm_shmem.v	
Syntax	`data:= shmem_read(addr, leng)`	
Arguments	`addr`	BFM shared memory starting address for reading data
	`leng`	Length, in bytes, of data read
Return	`data`	Data read from BFM shared memory. This parameter is implemented as a 64-bit vector. `leng` is 1- 8 bytes. If `leng` is less than 8 bytes, only the corresponding least significant bits of the returned data are valid. Bits 7 down to 0 are read from the location specified by `addr`; bits 15 down to 8 are read from the addr+1 location, etc.

13.5.10.4. shmem_display Verilog HDL Function

The `shmem_display` Verilog HDL function displays a block of data from the BFM shared memory.

Location	altrpcietb_bfm_shmem.v	
Syntax	Verilog HDL: `dummy_return:=shmem_display(addr, leng, word_size, flag_addr, msg_type);`	
Arguments	`addr`	BFM shared memory starting address for displaying data.
	`leng`	Length, in bytes, of data to display.

continued...

Location	altrpcietb_bfm_shmem.v	
	`word_size`	Size of the words to display. Groups individual bytes into words. Valid values are 1, 2, 4, and 8.
	`flag_addr`	Adds a <== flag to the end of the display line containing this address. Useful for marking specific data. Set to a value greater than 2**21 (size of BFM shared memory) to suppress the flag.
	`msg_type`	Specifies the message type to be displayed at the beginning of each line. See "BFM Log and Message Procedures" on page 18–37 for more information about message types. Set to one of the constants defined in Table 18–36 on page 18–41.

13.5.10.5. shmem_fill Procedure

The `shmem_fill` procedure fills a block of BFM shared memory with a specified data pattern.

Location	altrpcietb_bfm_shmem.v	
Syntax	`shmem_fill(addr, mode, leng, init)`	
Arguments	`addr`	BFM shared memory starting address for filling data.
	`mode`	Data pattern used for filling the data. Should be one of the constants defined in section *Shared Memory Constants*.
	`leng`	Length, in bytes, of data to fill. If the length is not a multiple of the incrementing data pattern width, then the last data pattern is truncated to fit.
	`init`	Initial data value used for incrementing data pattern modes. This argument is `reg [63:0]`. The necessary least significant bits are used for the data patterns that are smaller than 64 bits.

Related Information

Shared Memory Constants on page 138

13.5.10.6. shmem_chk_ok Function

The shmem_chk_ok function checks a block of BFM shared memory against a specified data pattern.

Location	altrpcietb_bfm_shmem.v	
Syntax	`result:= shmem_chk_ok(addr, mode, leng, init, display_error)`	
Arguments	`addr`	BFM shared memory starting address for checking data.
	`mode`	Data pattern used for checking the data. Should be one of the constants defined in section "Shared Memory Constants" on page 18–35.
	`leng`	Length, in bytes, of data to check.
	`init`	This argument is `reg [63:0]`.The necessary least significant bits are used for the data patterns that are smaller than 64-bits.
	`display_error`	When set to 1, this argument displays the data failing comparison on the simulator standard output.
Return	`Result`	Result is 1-bit. • 1'b1 — Data patterns compared successfully • 1'b0 — Data patterns did not compare successfully

13.5.11. BFM Log and Message Procedures

The following procedures and functions are available in the Verilog HDL include file `altpcietb_bfm_log.v`

These procedures provide support for displaying messages in a common format, suppressing informational messages, and stopping simulation on specific message types.

The following constants define the type of message and their values determine whether a message is displayed or simulation is stopped after a specific message. Each displayed message has a specific prefix, based on the message type in the following table.

You can suppress the display of certain message types. The default values determining whether a message type is displayed are defined in the following table. To change the default message display, modify the display default value with a procedure call to `ebfm_log_set_suppressed_msg_mask`.

Certain message types also stop simulation after the message is displayed. The following table shows the default value determining whether a message type stops simulation. You can specify whether simulation stops for particular messages with the procedure `ebfm_log_set_stop_on_msg_mask`.

All of these log message constants type `integer`.

Table 80. Log Messages

Constant (Message Type)	Description	Mask Bit No	Display by Default	Simulation Stops by Default	Message Prefix
EBFM_MSG_DEBUG	Specifies debug messages.	0	No	No	DEBUG:
EBFM_MSG_INFO	Specifies informational messages, such as configuration register values, starting and ending of tests.	1	Yes	No	INFO:
EBFM_MSG_WARNING	Specifies warning messages, such as tests being skipped due to the specific configuration.	2	Yes	No	WARNING:
EBFM_MSG_ERROR_INFO	Specifies additional information for an error. Use this message to display preliminary information before an error message that stops simulation.	3	Yes	No	ERROR:
EBFM_MSG_ERROR_CONTINUE	Specifies a recoverable error that allows simulation to continue. Use this error for data comparison failures.	4	Yes	No	ERROR:
EBFM_MSG_ERROR_FATAL	Specifies an error that stops simulation because the error leaves the testbench in a state where further simulation is not possible.	N/A	Yes Cannot suppress	Yes Cannot suppress	FATAL:
EBFM_MSG_ERROR_FATAL_TB_ERR	Used for BFM test driver or Root Port BFM fatal errors. Specifies an error that stops simulation because the error leaves the testbench in a	N/A	Y Cannot suppress	Y Cannot suppress	FATAL:

continued...

Constant (Message Type)	Description	Mask Bit No	Display by Default	Simulation Stops by Default	Message Prefix
	state where further simulation is not possible. Use this error message for errors that occur due to a problem in the BFM test driver module or the Root Port BFM, that are not caused by the Endpoint Application Layer being tested.				

13.5.11.1. ebfm_display Verilog HDL Function

The `ebfm_display` procedure or function displays a message of the specified type to the simulation standard output and also the log file if `ebfm_log_open` is called.

A message can be suppressed, simulation can be stopped or both based on the default settings of the message type and the value of the bit mask when each of the procedures listed below is called. You can call one or both of these procedures based on what messages you want displayed and whether or not you want simulation to stop for specific messages.

- When `ebfm_log_set_suppressed_msg_mask` is called, the display of the message might be suppressed based on the value of the bit mask.
- When `ebfm_log_set_stop_on_msg_mask` is called, the simulation can be stopped after the message is displayed, based on the value of the bit mask.

Location	altrpcietb_bfm_log.v	
Syntax	`Verilog HDL: dummy_return:=ebfm_display(msg_type, message);`	
Argument	`msg_type`	Message type for the message. Should be one of the constants defined in Table 79 on page 138.
	`message`	The message string is limited to a maximum of 100 characters. Also, because Verilog HDL does not allow variable length strings, this routine strips off leading characters of 8'h00 before displaying the message.
Return	`always 0`	Applies only to the Verilog HDL routine.

13.5.11.2. ebfm_log_stop_sim Verilog HDL Function

The `ebfm_log_stop_sim` procedure stops the simulation.

Location	altrpcietb_bfm_log.v	
Syntax	Verilog HDL: `return:=ebfm_log_stop_sim(success);`	
Argument	`success`	When set to a 1, this process stops the simulation with a message indicating successful completion. The message is prefixed with SUCCESS. Otherwise, this process stops the simulation with a message indicating unsuccessful completion. The message is prefixed with FAILURE.
Return	Always 0	This value applies only to the Verilog HDL function.

13.5.11.3. ebfm_log_set_suppressed_msg_mask Task

The `ebfm_log_set_suppressed_msg_mask` procedure controls which message types are suppressed.

Location	altrpcietb_bfm_log.v	
Syntax	`ebfm_log_set_suppressed_msg_mask (msg_mask)`	
Argument	`msg_mask`	This argument is `reg [EBFM_MSG_ERROR_CONTINUE: EBFM_MSG_DEBUG]`. A 1 in a specific bit position of the `msg_mask` causes messages of the type corresponding to the bit position to be suppressed.

13.5.11.4. ebfm_log_set_stop_on_msg_mask Verilog HDL Task

The `ebfm_log_set_stop_on_msg_mask` procedure controls which message types stop simulation. This procedure alters the default behavior of the simulation when errors occur as described in the *BFM Log and Message Procedures*.

Location	altrpcietb_bfm_log.v	
Syntax	`ebfm_log_set_stop_on_msg_mask (msg_mask)`	
Argument	`msg_mask`	This argument is `reg [EBFM_MSG_ERROR_CONTINUE:EBFM_MSG_DEBUG]`. A 1 in a specific bit position of the `msg_mask` causes messages of the type corresponding to the bit position to stop the simulation after the message is displayed.

Related Information

BFM Log and Message Procedures on page 141

13.5.11.5. ebfm_log_open Verilog HDL Function

The `ebfm_log_open` procedure opens a log file of the specified name. All displayed messages are called by `ebfm_display` and are written to this log file as simulator standard output.

Location	altrpcietb_bfm_log.v	
Syntax	`ebfm_log_open (fn)`	
Argument	`fn`	This argument is type `string` and provides the file name of log file to be opened.

13.5.11.6. ebfm_log_close Verilog HDL Function

The `ebfm_log_close` procedure closes the log file opened by a previous call to `ebfm_log_open`.

Location	altrpcietb_bfm_log.v
Syntax	`ebfm_log_close`
Argument	NONE

13.5.12. Verilog HDL Formatting Functions

The Verilog HDL Formatting procedures and functions are available in the `altpcietb_bfm_driver_rp.v`. The formatting functions are only used by Verilog HDL. All these functions take one argument of a specified length and return a vector of a specified length.

13.5.12.1. himage1

This function creates a one-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_log.v	
Syntax	`string:= himage(vec)`	
Argument	`vec`	Input data type `reg` with a `range` of 3:0.
Return range	`string`	Returns a 1-digit hexadecimal representation of the input argument. Return data is type `reg` with a `range` of 8:1

13.5.12.2. himage2

This function creates a two-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_log.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 7:0.
Return range	`string`	Returns a 2-digit hexadecimal presentation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 16:1

13.5.12.3. himage4

This function creates a four-digit hexadecimal string representation of the input argument can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_log.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 15:0.
Return range	Returns a four-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 32:1.	

13.5.12.4. himage8

This function creates an 8-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v altpcietb_bfm_log.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type reg with a range of 31:0.
Return range	`string`	Returns an 8-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 64:1.

13.5.12.5. himage16

This function creates a 16-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_log.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type reg with a range of 63:0.
Return range	`string`	Returns a 16-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 128:1.

13.5.12.6. dimage1

This function creates a one-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_log.v	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 1-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 8:1. Returns the letter *U* if the value cannot be represented.

13.5.12.7. dimage2

This function creates a two-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_log.v	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 2-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 16:1. Returns the letter *U* if the value cannot be represented.

13.5.12.8. dimage3

This function creates a three-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_log.v	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	string	Returns a 3-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 24:1. Returns the letter *U* if the value cannot be represented.

13.5.12.9. dimage4

This function creates a four-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	**altpcietb_bfm_log.v**	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 4-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 32:1. Returns the letter *U* if the value cannot be represented.

13.5.12.10. dimage5

This function creates a five-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	**altpcietb_bfm_log.v**	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 5-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 40:1. Returns the letter *U* if the value cannot be represented.

13.5.12.11. dimage6

This function creates a six-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	**altpcietb_bfm_log.v**	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 6-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 48:1. Returns the letter *U* if the value cannot be represented.

13.5.12.12. dimage7

This function creates a seven-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	**altpcietb_bfm_log.v**	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 7-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 56:1. Returns the letter <*U*> if the value cannot be represented.

13.6. Setting Up Simulation

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

13.6.1. 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, `<`*`testbench_dir`*`>/pcie_<`*`dev`*`>_hip_avmm_bridge_example_design_tb/ip/pcie_example_design_tb/DUT_pcie_tb_ip/altera_pcie_<`*`dev`*`>_tbed_<`*`ver`*`>/sim/altpcie_<`*`dev`*`>_tbed_hwtcl.v`.
2. Open `altpcie_<`*`dev`*`>_tbed_hwtcl.v`.
3. Search for the string, `serial_sim_hwtcl`. Set the value of this parameter to 0 if it is 1.
4. Save `altpcie_<`*`dev`*`>_tbed_hwtcl.v`.

13.6.2. Viewing the Important PIPE Interface Signals

You can view the most important PIPE interface signals, `txdata`, `txdatak`, `rxdata`, and `rxdatak` at the following level of the design hierarchy: `altpcie_<`*`device`*`>_hip_pipen1b|twentynm_hssi_<`*`gen`*`>_<`*`lanes`*`>_pcie_hip`.

13.6.3. Disabling the Scrambler for Gen1 and Gen2 Simulations

The 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_<`*`dev`*`>_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`.

13.6.4. Disabling 8B/10B Encoding and Decoding for Gen1 and Gen2 Simulations

You can disable 8B/10B encoding and decoding to facilitate debugging.

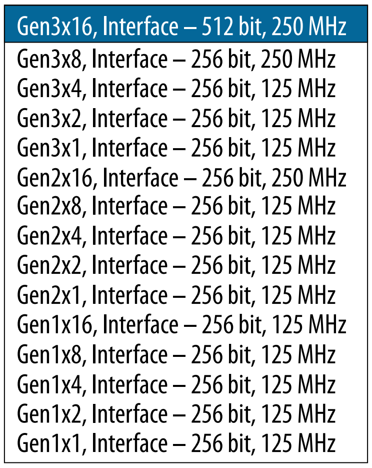
For Gen1 and Gen2 variants, you can disable 8B/10B encoding and decoding by setting `test_in[2]` in `<`*`testbench_dir`*`>/pcie_<`*`dev`*`>_hip_avmm_bridge_example_design_tb/ip/pcie_example_design_tb/DUT_pcie_tb_ip/altera_pcie_<`*`dev`*`>_tbed_<`*`ver`*`>/sim/altpcietb_bfm)top_rp.v`.

14. Avalon-MM Testbench and Design Example for Root Port

This chapter introduces the Root Port design example, which includes a testbench, a Bus Functional Model (BFM), and a test driver module.

Starting with the 18.0 release of Intel Quartus Prime, the Root Port design example is available for the following variants of the Intel Arria 10 Avalon-MM Hard IP for PCIe:

Figure 63. Avalon-MM Intel Arria 10 Hard IP for PCIe variants with an available Root Port design example



Gen3x16, Interface – 512 bit, 250 MHz
Gen3x8, Interface – 256 bit, 250 MHz
Gen3x4, Interface – 256 bit, 125 MHz
Gen3x2, Interface – 256 bit, 125 MHz
Gen3x1, Interface – 256 bit, 125 MHz
Gen2x16, Interface – 256 bit, 250 MHz
Gen2x8, Interface – 256 bit, 125 MHz
Gen2x4, Interface – 256 bit, 125 MHz
Gen2x2, Interface – 256 bit, 125 MHz
Gen2x1, Interface – 256 bit, 125 MHz
Gen1x16, Interface – 256 bit, 125 MHz
Gen1x8, Interface – 256 bit, 125 MHz
Gen1x4, Interface – 256 bit, 125 MHz
Gen1x2, Interface – 256 bit, 125 MHz
Gen1x1, Interface – 256 bit, 125 MHz

To generate an Avalon-MM Root Port design example, configure the IP Core as a Root Port and select the **Avalon-MM** application interface type. For more details, refer to the following *Example Design Generation* section.

The simulation testbench instantiates a Root Port design example of the Avalon-MM Intel Arria 10 Hard IP for PCIe and an Endpoint BFM, which sets up all the basic configuration registers in the Root Port. This configuration allows the Root Port to initiate link training and bus enumeration.

You can compile the Root Port design example to generate the `.sof` file, which you can program into your FPGA device to perform board-level hardware tests. For simulation, the Root Port design example uses a JTAG master bridge BFM to configure the Root Port and initiate link training and bus enumeration. The JTAG master bridge BFM can also drive the TXS Avalon-MM interface to perform memory reads and writes.

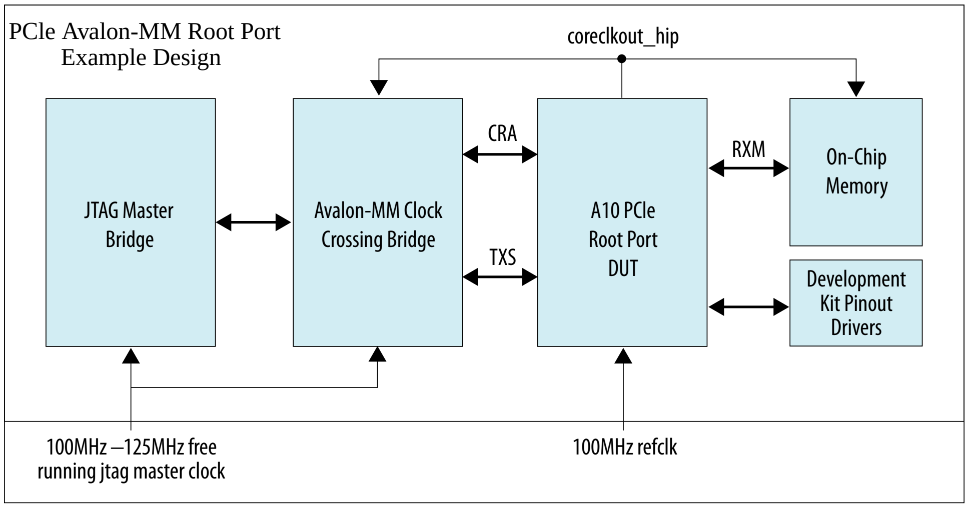
The testbench and Root Port design example provide a simple method to do basic testing of the application layer logic that interfaces with the IP Core. The Endpoint BFM allows you to create and run simple task stimuli with configurable parameters to exercise the basic functionality of the design example. The testbench and BFM are not intended to be a substitute for a full verification environment, and do not cover corner cases and certain traffic profiles. To ensure the best verification coverage possible, Intel recommends that you obtain commercially available PCI Express verification IP and tools to run simulations, or do your own extensive hardware testing, or both.

14.1. Overview of the Design Example

The Avalon-MM Root Port design example includes the following modules:

1. Intel Arria 10 PCIe Root Port DUT
2. On-chip memory for RXM_BAR0
3. Avalon-MM clock-crossing bridge
4. JTAG master bridge
5. Small module for convenient pin assignments for the development kit

Figure 64. PCIe Avalon-MM Root Port Design Example



After the design example is generated, a `pcie_example_design.qpf` and `pcie_example_design.qsf` files are generated along with a `pcie_example_design.qsys` file as shown in the directory structure. You can open the `pcie_example_design.qpf` project file in Intel Quartus Prime and directly run compilation.

14.1.1. Example Design Generation

To generate an Intel Arria 10 Root Port design example, do the following steps:

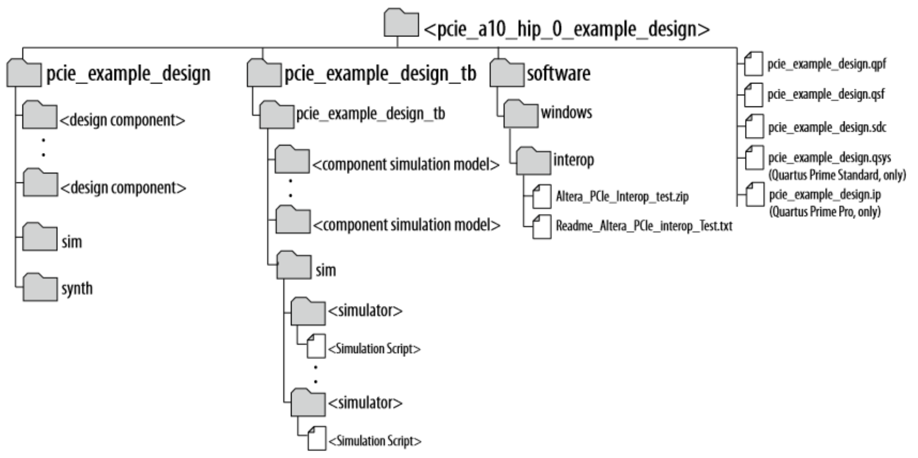
1. Open Platform Designer.
2. In the Platform Designer IP Catalog, select **Intel Arria 10/Cyclone 10 Hard IP for PCI Express**.
3. After the `altera_pcie_a10_hip` IP GUI opens, select the following parameters in the **System Settings** tab under **IP Settings**:

   a. For **Application interface type**, select **Avalon-MM**.
   b. For **Hard IP mode**, select any variant.
   c. For **Port type**, select **Root port**.
4. In the **Base Address Registers** tab, only enable BAR0, or BAR0 and BAR1. All other BARs are disabled in the current Root Port design example.
   a. If you set BAR0 to use **64-bit prefetchable memory**, you need to disable BAR1.
   b. If you set BAR0 to use **32-bit prefetchable memory** or **32-bit non-prefetchable memory**, you can enable or disable BAR1.
   c. The BAR0 and BAR1 sizes are not configurable in the current Intel Arria 10 PCIe IP GUI. After you generate the design example, you can open it using Platform Designer and reselect the memory size. The BAR size is then automatically updated.
5. Click the **Generate Example Design** button. A small window appears allowing you to select the directory to generate the Root Port design example, and give a name to the design example.
6. Click **OK** in the **Select Example Design Directory** window to let Platform Designer generate a design example for you.

14.1.2. Directory Structure for the Generated Design Example

Below is the directory structure for the generated Root Port design example:

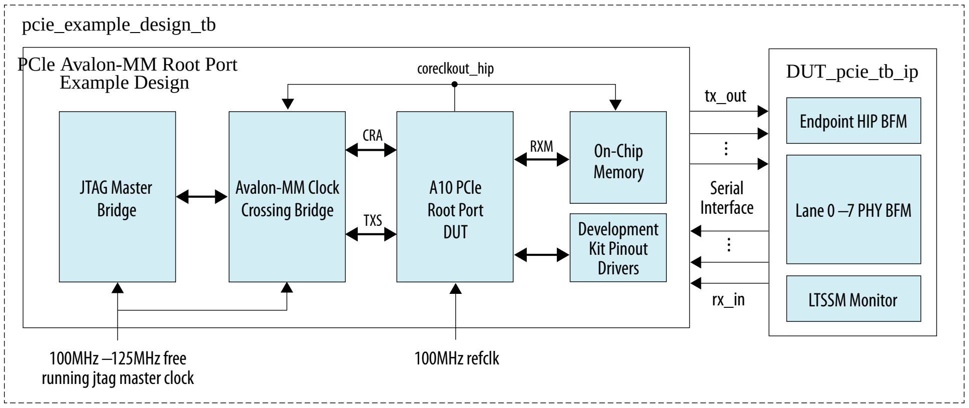
Figure 65. Directory Structure for the Design Example



14.2. Overview of the Simulation Testbench

The Intel Arria 10 Avalon-MM Root Port design example only supports serial mode simulations. The Root Port DUT and Endpoint BFM communicate via high-speed serial links. Below is the architecture of the `pcie_example_design_tb` simulation environment.

Figure 66. PCIe Avalon-MM Root Port Simulation Environment



14.2.1. Simulating the Design Example

Below are the steps to run a simulation:

1. Go to the testbench simulation directory.
2. Run the simulation script for the simulator of your choice. Refer to the table below for more details.
3. Analyze the results.

Table 81. Steps to Run Simulation

Simulator	Working Directory	Instructions
ModelSim	`<example_design>/ pcie_example_design_tb/ pcie_example_design_tb/sim/mentor/`	1. Invoke vsim 2. `do msim_setup.tcl` 3. `ld_debug` 4. `run -all` 5. A successful simulation ends with the following message, "Simulation passed"
VCS	`<example_design>/ pcie_example_design_tb/ pcie_example_design_tb/sim/ synopsys/vcs`	1. sh vcs_setup.sh USER_DEFINED_SIM_OPTIONS="" 2. A successful simulation ends with the following message, "Simulation passed"
NCSim	`<example_design>/ pcie_example_design_tb/ pcie_example_design_tb/sim/cadence`	1. sh ncsim_setup.sh USER_DEFINED_SIM_OPTIONS="" 2. A successful simulation ends with the following message, "Simulation passed"
Xcelium* Parallel Simulator	`<example_design>/ pcie_example_design_tb/ pcie_example_design_tb/sim/xcelium`	1. sh xcelium_setup.sh USER_DEFINED_SIM_OPTIONS="" USER_DEFINED_ELAB_OPTIONS="-NOWARN\ CSINFI" 2. A successful simulation ends with the following message, "Simulation passed"

Following is the partial transcript from a successful simulation of the Intel Arria 10 Avalon-MM Root Port design example:

```
INFO:              865 ns   EP Link Speed change to:              Gen1
INFO:             4205 ns   EP LTSSM State: DETECT.ACTIVE
INFO:             5309 ns   EP LTSSM State: POLLING.ACTIVE
INFO:            18173 ns   EP LTSSM State: DETECT.QUIET
```

```
INFO:           18509 ns  EP LTSSM State: DETECT.ACTIVE
INFO:           19549 ns  EP LTSSM State: POLLING.ACTIVE
INFO:           20349 ns  RP Link Speed change to:              Gen1
INFO:           20377 ns  RP Link Speed change to:              0
INFO:           20605 ns  RP Link Speed change to:              Gen1
INFO:           23965 ns  RP LTSSM State: DETECT.ACTIVE
INFO:           28765 ns  RP LTSSM State: DETECT.QUIET
INFO:           32029 ns  RP LTSSM State: DETECT.ACTIVE
INFO:           32413 ns  EP LTSSM State: DETECT.QUIET
INFO:           35693 ns  EP LTSSM State: DETECT.ACTIVE
INFO:           36765 ns  EP LTSSM State: POLLING.ACTIVE
INFO:           36797 ns  RP LTSSM State: POLLING.ACTIVE
INFO:           39901 ns  EP LTSSM State: POLLING.CONFIG
INFO:           40317 ns  RP LTSSM State: POLLING.CONFIG
INFO:           41597 ns  RP LTSSM State: CONFIG.LINKWIDTH.START
INFO:           41821 ns  EP LTSSM State: CONFIG.LINKWIDTH.START
INFO:           42237 ns  EP LTSSM State: CONFIG.LINKWIDTH.ACCEPT
INFO:           42781 ns  RP LTSSM State: CONFIG.LINKWIDTH.ACCEPT
INFO:           43165 ns  RP LTSSM State: CONFIG.LANENUM.WAIT
INFO:           43709 ns  EP LTSSM State: CONFIG.LANENUM.WAIT
INFO:           44029 ns  EP LTSSM State: CONFIG.LANENUM.ACCEPT
INFO:           44189 ns  RP LTSSM State: CONFIG.LANENUM.ACCEPT
INFO:           44573 ns  RP LTSSM State: CONFIG.COMPLETE
INFO:        Start Enumeration Process
INFO:           45117 ns  EP LTSSM State: CONFIG.COMPLETE
INFO:           46621 ns  RP LTSSM State: CONFIG.IDLE
INFO:           47389 ns  EP LTSSM State: CONFIG.IDLE
INFO:           47581 ns  EP LTSSM State: L0
INFO:           47805 ns  RP LTSSM State: L0
INFO:           75005 ns  RP LTSSM State: RECOVERY.RCVRLOCK
INFO:           75869 ns  EP LTSSM State: RECOVERY.RCVRLOCK
INFO:           76637 ns  EP LTSSM State: RECOVERY.RCVRCFG
INFO:           78045 ns  RP LTSSM State: RECOVERY.RCVRCFG
INFO:           80285 ns  RP LTSSM State: RECOVERY.SPEED
INFO:           80509 ns  EP LTSSM State: RECOVERY.SPEED
INFO:           82345 ns  RP Link Speed change to:              Gen3
INFO:           82353 ns  RP LTSSM State: RECOVERY.RCVRLOCK
INFO:           82389 ns  EP Link Speed change to:              Gen3
INFO:           82397 ns  EP LTSSM State: RECOVERY.RCVRLOCK
INFO:           82933 ns  EP LTSSM State: RECOVERY.RCVRCFG
INFO:           83769 ns  RP LTSSM State: RECOVERY.RCVRCFG
INFO:           84377 ns  RP LTSSM State: RECOVERY.IDLE
INFO:           84997 ns  EP LTSSM State: RECOVERY.IDLE
INFO:           85093 ns  EP LTSSM State: L0
INFO:           85193 ns  RP LTSSM State: L0
INFO:           82345 ns  RP Link Speed change to:              Gen3
INFO:           82353 ns  RP LTSSM State: RECOVERY.RCVRLOCK
INFO:           82389 ns  EP Link Speed change to:              Gen3
INFO:           82397 ns  EP LTSSM State: RECOVERY.RCVRLOCK
INFO:           82933 ns  EP LTSSM State: RECOVERY.RCVRCFG
INFO:           83769 ns  RP LTSSM State: RECOVERY.RCVRCFG
INFO:           84377 ns  RP LTSSM State: RECOVERY.IDLE
INFO:           84997 ns  EP LTSSM State: RECOVERY.IDLE
INFO:           85093 ns  EP LTSSM State: L0
INFO:           85193 ns  RP LTSSM State: L0
INFO:        Finish Enumeration Process
TXS interface sent 12345678, received  12345678
TXS interface sent 89abcdef, received  89abcdef
TXS interface sent 5f5f5f5f, received  5f5f5f5f
TXS interface sent c1c1c1c1, received  c1c1c1c1
Simulation pass
```

Send Feedback

15. Debugging

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

15.1. Simulation Fails To Progress Beyond Polling.Active State

If your PIPE simulation cycles between the Detect.Quiet, Detect.Active, and Polling.Active LTSSM states, the PIPE interface width may be incorrect.

Make the changes shown in the following table for the 32-bit PIPE interface.

Table 82. Changes for 32-Bit PIPE Interface

8-Bit PIPE Interface	32-Bit PIPE Interface
`output wire [7:0] pcie_a10_hip_0_hip_pipe_txdata0`	`output wire [31:0] pcie_a10_hip_0_hip_pipe_txdata0`
`input wire [7:0] pcie_a10_hip_0_hip_pipe_rxdata0`	`input wire [31:0] pcie_a10_hip_0_hip_pipe_rxdata0`
`output wire pcie_a10_simulation_inst_pcie_a10_hip_0_hip_pipe_txdatak0`	`output wire [3:0] pcie_a10_simulation_inst_pcie_a10_hip_0_hip_pipe_txdatak0`
`input wire pcie_a10_simulation_inst_pcie_a10_hip_0_hip_pipe_rxdatak0`	`input wire [3:0] pcie_a10_simulation_inst_pcie_a10_hip_0_hip_pipe_rxdatak0`

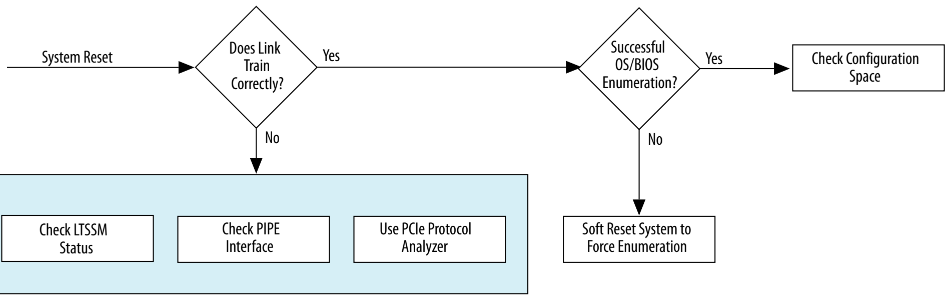
15.2. Hardware Bring-Up Issues

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. Intel recommends a systematic approach to diagnosing bring-up issues as illustrated in the following figure.

Figure 67. Debugging Link Training Issues


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

15.3. 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:

- Signal Tap Embedded Logic Analyzer
- Third-party PCIe protocol analyzer

You can use Signal Tap Embedded Logic Analyzer to diagnose the LTSSM state transitions that are occurring on the PIPE interface. The `ltssmstate` 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, 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` to determine the cause.

Related Information

Reset, Status, and Link Training Signals on page 52

15.4. Creating a Signal Tap Debug File to Match Your Design Hierarchy

For Intel Arria 10 and Intel Cyclone 10 GX devices, the Intel Quartus Prime software generates two files, `build_stp.tcl` and `<`*`ip_core_name`*`>.xml`. You can use these files to generate a Signal Tap file with probe points matching your design hierarchy.

The Intel Quartus Prime software stores these files in the `<`*`IP core directory`*`>/synth/debug/stp/` directory.

Synthesize your design using the Intel Quartus Prime software.

1. To open the Tcl console, click **View ➤ Utility Windows ➤ Tcl Console**.
2. Type the following command in the Tcl console:
   `source <`*`IP core directory`*`>/synth/debug/stp/build_stp.tcl`
3. To generate the STP file, type the following command:

```
main -stp_file <output stp file name>.stp -xml_file <input
xml_file name>.xml -mode build
```

4. To add this Signal Tap file (**.stp**) to your project, select **Project ➤ Add/Remove Files in Project**. Then, compile your design.
5. To program the FPGA, click **Tools ➤ Programmer**.
6. To start the Signal Tap Logic Analyzer, click **Quartus Prime ➤ Tools ➤ Signal Tap Logic Analyzer**.

   The software generation script may not assign the Signal Tap acquisition clock in `<output stp file name>.stp`. Consequently, the Intel Quartus Prime software automatically creates a clock pin called `auto_stp_external_clock`. You may need to manually substitute the appropriate clock signal as the Signal Tap sampling clock for each STP instance.
7. Recompile your design.
8. To observe the state of your IP core, click **Run Analysis**.

   You may see signals or Signal Tap instances that are red, indicating they are not available in your design. In most cases, you can safely ignore these signals and instances. They are present because software generates wider buses and some instances that your design does not include.

15.5. Use Third-Party PCIe Analyzer

A third-party protocol 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 protocol 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.

15.6. BIOS Enumeration Issues

Both FPGA programming (configuration) and the initialization of a PCIe link require time. Potentially, an Intel 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 enumeration, the OS does not include the Hard IP for PCI Express in its device map.

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 enumeration.

A. PCI Express Protocol Stack

The Avalon-MM Intel Arria 10 or Intel Cyclone 10 GX 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.

Figure 68. Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express Using the Avalon-MM Interface

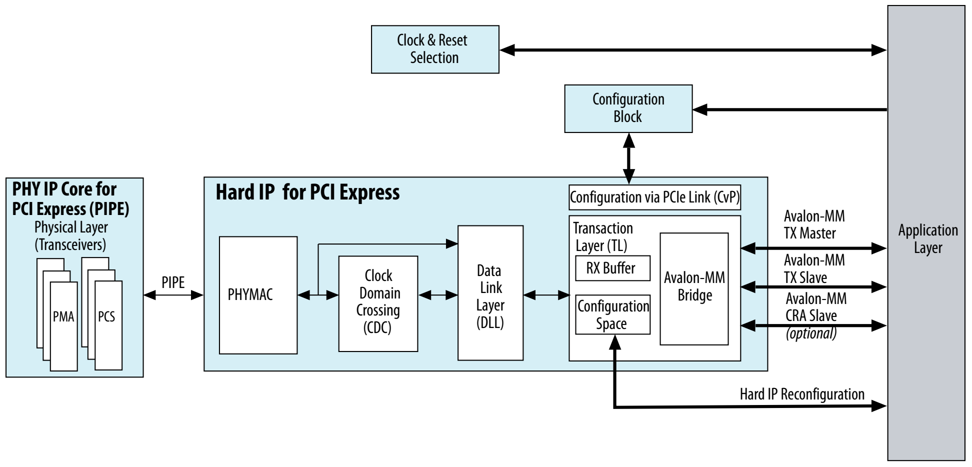


Table 83. Application Layer Clock Frequencies

Lanes	Gen1	Gen2	Gen3
×1	125 MHz @ 64 bits or 62.5 MHz @ 64 bits	125 MHz @ 64 bits	125 MHz @64 bits
×2	125 MHz @ 64 bits	125 MHz @ 128 bits	250 MHz @ 64 bits or 125 MHz @ 128 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

A.1. Top-Level Interfaces

A.1.1. 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.

Avalon-MM slaves use byte addresses. A slave only accepts addresses that are a multiple of its data width. Consequently, the lowest 2 bits of 32-bit address must be zero. Byte enables allow partial word access. For example, a write of 2 bytes at address 2 would have 4’b1100 for the byte enables. For larger accesses, additional low-order bits are unused, as shown in the following table.

Table 84. Avalon-MM Address Bits used for 32-, 64-, 128- and 256-Bit Data Widths

Data Width	Address Bits Used	Address Bits Set to 0 and Ignored
32 bits	addr[31:2]	addr[1:0]
64 bits	addr[63:3]	addr[2:0]
128 bits	addr[63:4]	addr[3:0]
256 bits	addr[63:5]	addr[4:0]

Related Information

- 64- or 128-Bit Avalon-MM Interface to the Endpoint Application Layer on page 45
- Avalon Interface Specifications

A.1.2. 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, IP core includes an embedded hard reset controller. This reset controller exits the reset state after the periphery of the device is initialized.

A.1.3. Interrupts

The Hard IP for PCI Express offers the following interrupt mechanisms:

- Message Signaled Interrupts (MSI)— MSI uses the TLP single dword memory writes to to implement interrupts. This interrupt mechanism conserves pins because it does not use separate wires for interrupts. In addition, the single dword provides flexibility in data presented in the interrupt message. The MSI Capability structure is stored in the Configuration Space and is programmed using Configuration Space accesses.
- MSI-X—The Transaction Layer generates MSI-X messages which are single dword memory writes. The MSI-X Capability structure points to an MSI-X table structure and MSI-X PBA structure which are stored in memory. This scheme is in contrast to the MSI capability structure, which contains all of the control and status information for the interrupt vectors.

Related Information

Interrupts for Endpoints when Multiple MSI/MSI-X Support Is Enabled on page 53

A.1.4. 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 simulation models support PIPE and serial simulation.
- For Gen3, the Intel BFM bypasses Gen3 Phase 2 and Phase 3 Equalization. However, Gen3 variants can perform Phase 2 and Phase 3 equalization if instructed by a third-party BFM.

Related Information

PIPE Interface Signals on page 59

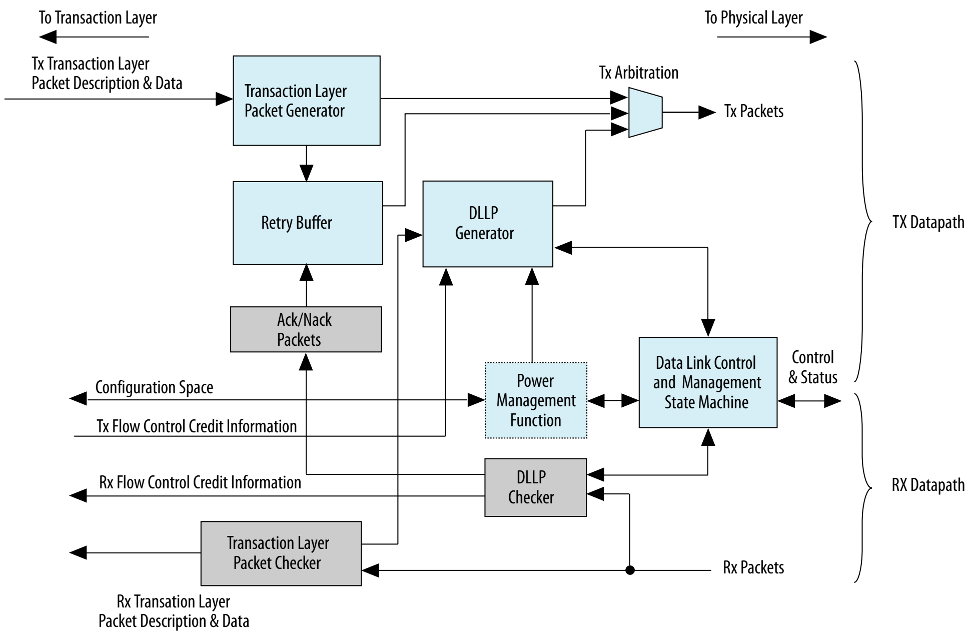
A.2. Data Link Layer

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.

The DLL implements the following functions:

- Link management through the reception and transmission of DLL Packets (DLLP), which are used for the following functions:
  - Power management of DLLP reception and transmission
  - To transmit and receive ACK/NAK packets
  - Data integrity through generation and checking of CRCs for TLPs and DLLPs
  - TLP retransmission in case of NAK DLLP reception or replay timeout, using the retry (replay) 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 69. Data Link Layer


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

The DLL has the following sub-blocks:

- Data Link Control and Management State Machine—This state machine connects to both the Physical Layer's LTSSM state machine and the Transaction Layer. It initializes the link and flow control credits and reports status to the Transaction Layer.
- 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. All of the Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCIe IP core variants do not support low power modes.
- 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, including a sequence number and a 32-bit Link 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. In case of 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)

A.3. Physical Layer

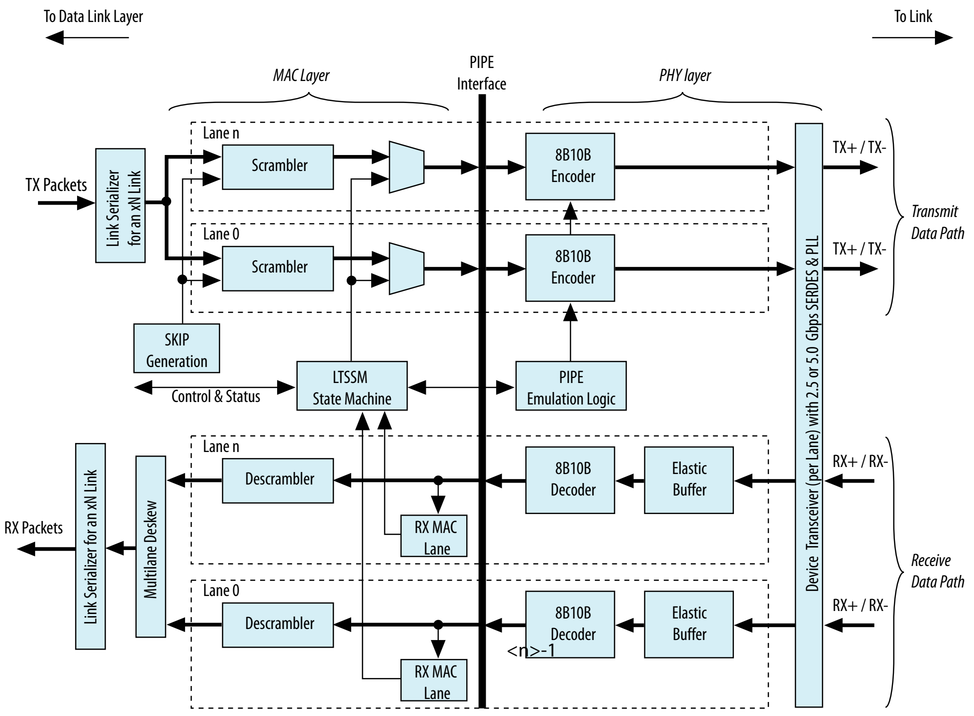
The Physical Layer is the lowest level of the PCI Express protocol stack. It is the layer closest to the serial link. It encodes and transmits packets across a link and accepts and decodes received packets. The Physical Layer connects to the link through a high-speed SERDES interface running at 2.5 Gbps for Gen1 implementations, at 2.5 or 5.0 Gbps for Gen2 implementations, and at 2.5, 5.0 or 8.0 Gbps for Gen3 implementations.

The Physical Layer is responsible for the following actions:

- Training the link
- Scrambling/descrambling and 8B/10B encoding/decoding for 2.5 Gbps (Gen1), 5.0 Gbps (Gen2), or 128b/130b encoding/decoding of 8.0 Gbps (Gen3) per lane
- Scrambling/descrambling and 8B/10B encoding/decoding for 2.5 Gbps (Gen1) and 5.0 Gbps (Gen2) per lane
- Serializing and deserializing data
- Equalization (Gen3)
- Operating the PIPE 3.0 Interface
- Implementing auto speed negotiation (Gen2 and Gen3)
- Implementing auto speed negotiation (Gen2)

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

Figure 70. Physical Layer Architecture



PHY Layer—The PHY layer includes the 8B/10B encode and decode functions for Gen1 and Gen2. The PHY also includes elastic buffering and serialization/deserialization functions.

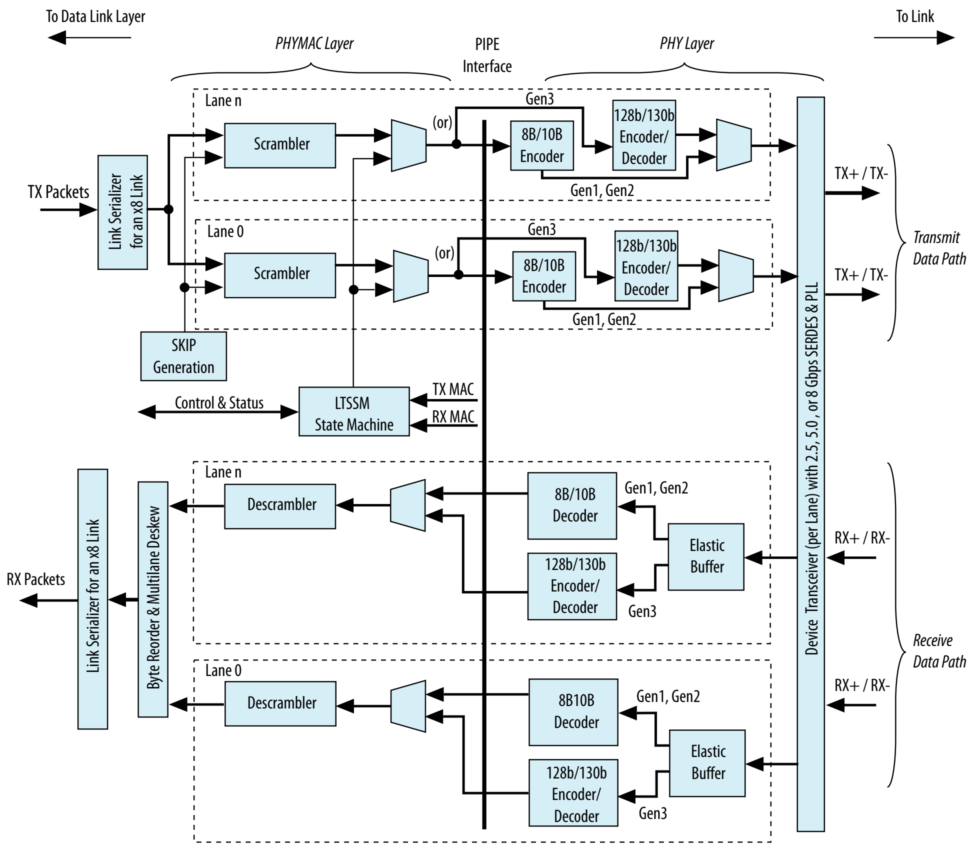
The Physical Layer is subdivided by the PIPE Interface Specification into two layers (bracketed horizontally in above figure):

- PHYMAC—The MAC layer includes the LTSSM and the scrambling/descrambling. byte reordering, and multilane deskew functions.
- Media Access Controller (MAC) Layer—The MAC layer includes the LTSSM and the scrambling and descrambling and multilane deskew functions.
- PHY Layer—The PHY layer includes the 8B/10B encode and decode functions for Gen1 and Gen2. It includes 128b/130b encode and decode functions for Gen3. The PHY also includes elastic buffering and serialization/deserialization functions.
- PHY Layer—The PHY layer includes the 8B/10B encode and decode functions for Gen1 and Gen2. The PHY also includes elastic buffering and serialization/ deserialization functions.

The Physical Layer integrates both digital and analog elements. Intel designed the PIPE interface to separate the PHYMAC from the PHY. The Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express complies with the PIPE interface specification.

Note: The internal PIPE interface is visible for simulation. It is not available for debugging in hardware using a logic analyzer such as Signal Tap. If you try to connect Signal Tap to this interface the design fails compilation.

Figure 71. Physical Layer Architecture



The PHYMAC block comprises 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 Ordered Set and SKP sub-block (LTSTX). 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 TX and RX training sequences 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 initialized lanes and the datapath. The multilane deskew implements an eight-word FIFO buffer 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.

A.4. 32-Bit PCI Express Avalon-MM Bridge

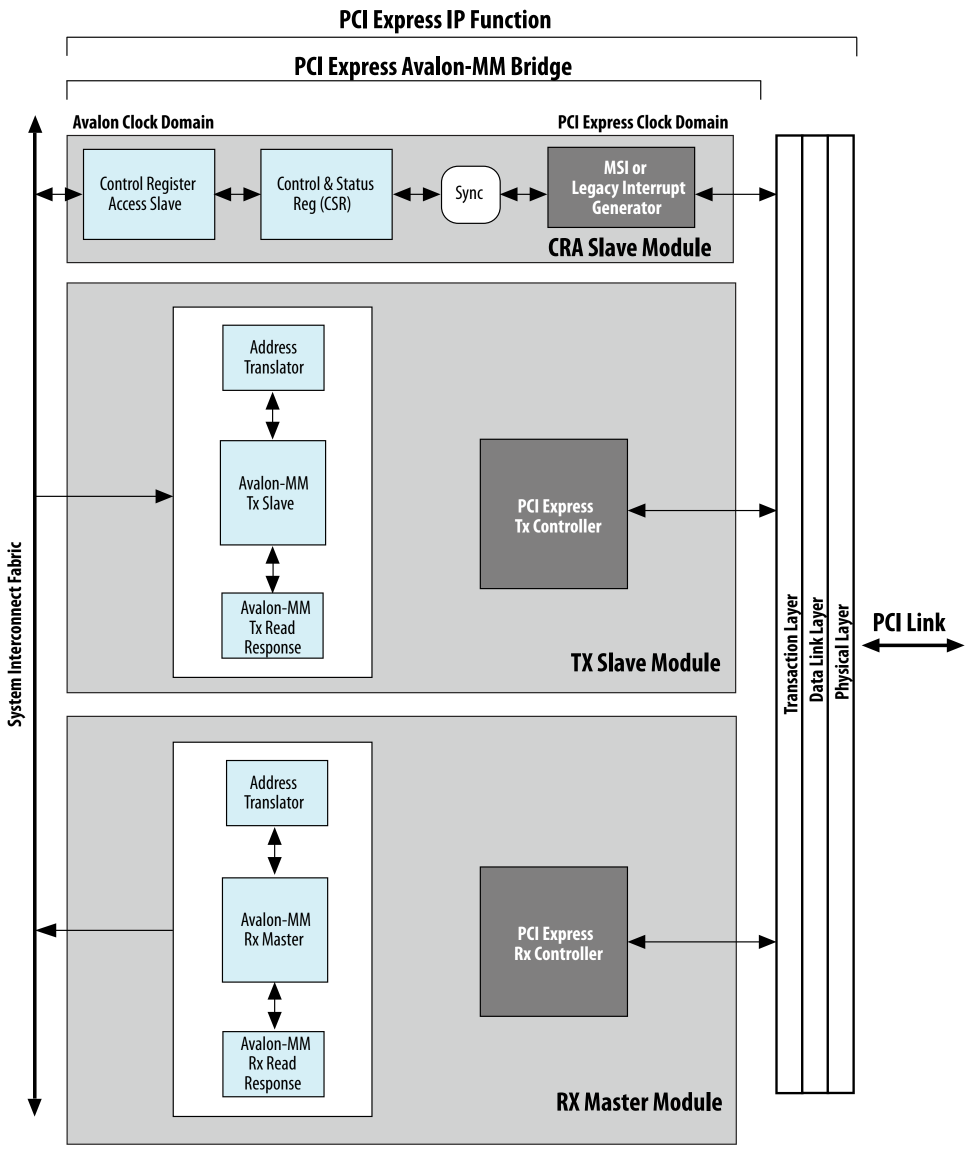
The Avalon-MM Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express includes an Avalon-MM bridge module that connects the Hard IP to the interconnect fabric. The bridge facilitates the design of Endpoints and Root Ports that include Platform Designer components.

The 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 KB 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** for the Avalon-MM Hard IP for PCI Express, Platform Designer substitutes an 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 72. PCI Express Avalon-MM Bridge


PCI Express IP Function
PCI Express Avalon-MM Bridge
Avalon Clock Domain
PCI Express Clock Domain
Control Register Access Slave
Control & Status Reg (CSR)
Sync
MSI or Legacy Interrupt Generator
CRA Slave Module
Address Translator
Avalon-MM Tx Slave
Avalon-MM Tx Read Response
PCI Express Tx Controller
TX Slave Module
Address Translator
Avalon-MM Rx Master
Avalon-MM Rx Read Response
PCI Express Rx Controller
RX Master Module
System Interconnect Fabric
Transaction Layer
Data Link Layer
Physical Layer
PCI Link

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 forward the address to a single RX Avalon-MM master port.

Related Information

Avalon-MM RX Master Block on page 176

A.4.1. 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.

A.4.2. 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 KB 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, Intel recommends using an Avalon-MM burst master without any byte-enable restrictions.

A.4.3. 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 KB 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 `txs_waitrequest` 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. If more than 1 dword is enabled, the enabled dwords must be contiguous. The following patterns are legal:

- 16'hF000
- 16'h0F00
- 16'h00F0
- 16'h000F
- 16'hFF00
- 16'h0FF0
- 16'h00FF
- 16'hFFF0
- 16'h0FFF
- 16'hFFFF

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.

A.4.4. 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.

A.4.5. 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 85. 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 Intel Arria 10 or Intel Cyclone 10 GX 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.

A.4.6. 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 parameter editor. Unsupported read requests generate a completer abort response.

Related Information

Minimizing BAR Sizes and the PCIe Address Space on page 171

A.4.7. 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 even to different BARs. You can specify the **Maximum payload size** parameter on the **Device** tab under the **PCI Express/PCI Capabilities** heading in the parameter editor.

Related Information

Device Capabilities on page 29

A.4.8. PCI Express-to-Avalon-MM Address Translation for 32-Bit Bridge

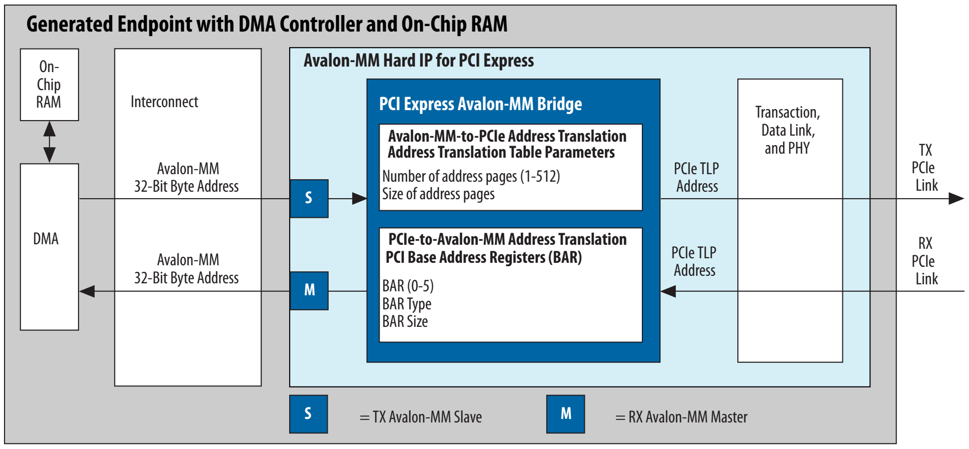
The PCI Express Avalon-MM bridge translates the system-level physical addresses, typically up to 64 bits, to the significantly smaller addresses required by the Application Layer's Avalon-MM slave components.

Note: Starting with the 13.0 version of the Quartus Prime 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 parameter editor.

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 73. Address Translation in TX and RX Directions For Endpoints


Generated Endpoint with DMA Controller and On-Chip RAM
On-Chip RAM
DMA
Interconnect
Avalon-MM 32-Bit Byte Address
Avalon-MM 32-Bit Byte Address
Avalon-MM Hard IP for PCI Express
PCI Express Avalon-MM Bridge
Avalon-MM-to-PCIe Address Translation
Address Translation Table Parameters
Number of address pages (1-512)
Size of address pages
PCIe-to-Avalon-MM Address Translation
PCI Base Address Registers (BAR)
BAR (0-5)
BAR Type
BAR Size
S
M
PCIe TLP Address
PCIe TLP Address
Transaction, Data Link, and PHY
TX PCIe Link
RX PCIe Link
S = TX Avalon-MM Slave
M = RX Avalon-MM Master

Note: When configured as a Root Port, a single RX Avalon-MM master forwards all RX TLPs to the Platform Designer 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 Platform Designer 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 parameter editor:

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 of 0x0000123456789000
3. A TLP received with address 0x0000123456789870
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

Avalon-MM-to-PCI Express Address Translation Algorithm for 32-Bit Addressing on page 173

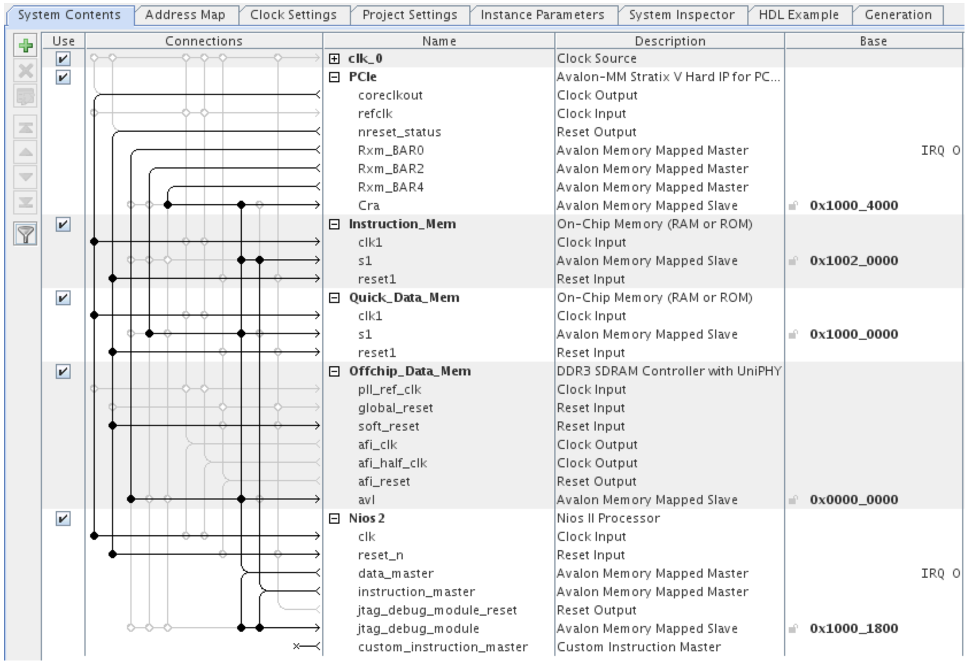
A.4.9. Minimizing BAR Sizes and the PCIe Address Space

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

- **Offchip_Data_Mem DDR3** (SDRAM Controller with UniPHY) controlling 256 MB of memory—Platform Designer auto-assigned a base address of 0x00000000
- **Quick_Data_Mem** (On-Chip Memory (RAM or ROM)) of 4 KB—Platform Designer auto-assigned a base address of 0x10000000
- **Instruction_Mem** (On-Chip Memory (RAM or ROM)) of 64 KB—Platform Designer auto-assigned a base address of 0x10020000
- **PCIe** (Avalon-MM Intel Arria 10 or Intel Cyclone 10 GX 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_module**
  - **instruction_master** connects to **Instruction_Mem s1**

Figure 74. Platform Designer System for PCI Express with Poor Address Space Utilization

The following figure uses a filter to hide the Conduit interfaces that are not relevant in this discussion.

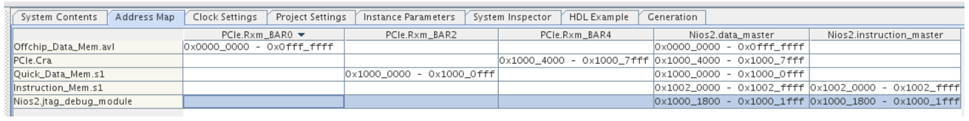


System Contents | Address Map | Clock Settings | Project Settings | Instance Parameters | System Inspector | HDL Example | Generation

Use	Name	Description	Base
✓	clk_0	Clock Source	
✓	PCIe	Avalon-MM Stratix V Hard IP for PC...	
	coreclkout	Clock Output	
	refclk	Clock Input	
	nreset_status	Reset Output	
	Rxm_BAR0	Avalon Memory Mapped Master	IRQ 0
	Rxm_BAR2	Avalon Memory Mapped Master	
	Rxm_BAR4	Avalon Memory Mapped Master	
	Cra	Avalon Memory Mapped Slave	0x1000_4000
✓	Instruction_Mem	On-Chip Memory (RAM or ROM)	
	clk1	Clock Input	
	s1	Avalon Memory Mapped Slave	0x1002_0000
	reset1	Reset Input	
✓	Quick_Data_Mem	On-Chip Memory (RAM or ROM)	
	clk1	Clock Input	
	s1	Avalon Memory Mapped Slave	0x1000_0000
	reset1	Reset Input	
✓	Offchip_Data_Mem	DDR3 SDRAM Controller with UniPHY	
	pll_ref_clk	Clock Input	
	global_reset	Reset Input	
	soft_reset	Reset Input	
	afi_clk	Clock Output	
	afi_half_clk	Clock Output	
	afi_reset	Reset Output	
	avl	Avalon Memory Mapped Slave	0x0000_0000
✓	Nios 2	Nios II Processor	
	clk	Clock Input	
	reset_n	Reset Input	
	data_master	Avalon Memory Mapped Master	IRQ 0
	instruction_master	Avalon Memory Mapped Master	
	jtag_debug_module_reset	Reset Output	
	jtag_debug_module	Avalon Memory Mapped Slave	0x1000_1800
	custom_instruction_master	Custom Instruction Master	

Figure 75. Poor Address Map

The following figure illustrates the address map for this system.



System Contents | Address Map | Clock Settings | Project Settings | Instance Parameters | System Inspector | HDL Example | Generation

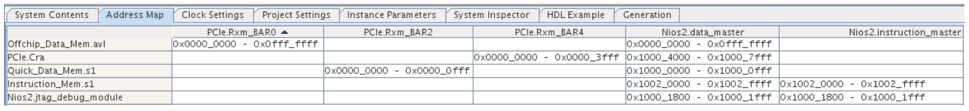
	PCIe.Rxm_BAR0	PCIe.Rxm_BAR2	PCIe.Rxm_BAR4	Nios2.data_master	Nios2.instruction_master
Offchip_Data_Mem.avl	0x0000_0000 - 0x0fff_ffff			0x0000_0000 - 0x0fff_ffff	
PCIe.Cra			0x1000_4000 - 0x1000_7fff	0x1000_4000 - 0x1000_7fff	
Quick_Data_Mem.s1		0x1000_0000 - 0x1000_0fff		0x1000_0000 - 0x1000_0fff	
Instruction_Mem.s1				0x1002_0000 - 0x1002_ffff	0x1002_0000 - 0x1002_ffff
Nios2.jtag_debug_module				0x1000_1800 - 0x1000_1fff	0x1000_1800 - 0x1000_1fff

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 KB. 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** is 16 KB. It should only require 14 address bits; however, it is also consuming 512 MB of address space.

This design is consuming 1.25 GB of PCIe address space when only 276 MB 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 76. Optimized Address Map



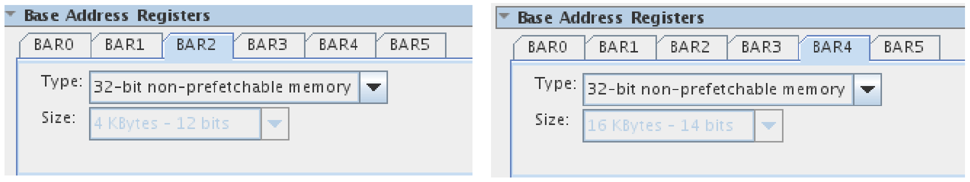
System Contents | Address Map | Clock Settings | Project Settings | Instance Parameters | System Inspector | HDL Example | Generation

	PCIe.Rxm_BAR0	PCIe.Rxm_BAR2	PCIe.Rxm_BAR4	Nios2.data_master	Nios2.instruction_master
Offchip_Data_Mem.avl	0x0000_0000 - 0x0fff_ffff			0x0000_0000 - 0x0fff_ffff	
PCIe.Cra			0x0000_0000 - 0x0000_3fff	0x1000_4000 - 0x1000_7fff	
Quick_Data_Mem.s1		0x0000_0000 - 0x0000_0fff		0x1000_0000 - 0x1000_0fff	
Instruction_Mem.s1				0x1002_0000 - 0x1002_ffff	0x1002_0000 - 0x1002_ffff
Nios2.jtag_debug_module				0x1000_1800 - 0x1000_1fff	0x1000_1800 - 0x1000_1fff

Send Feedback

Figure 77. 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.



Type: 32-bit non-prefetchable memory
Size: 4 KBytes - 12 bits

Type: 32-bit non-prefetchable memory
Size: 16 KBytes - 14 bits

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.

A.4.10. 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 KB to 4 GB when you customize your Avalon-MM Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express as described in *Avalon to PCIe Address Translation Settings* . 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 MB**
- **PCI Express Address Size—64 bits**

then the values in the following figure are:

- *N* = 20 (due to the 1 MB 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:

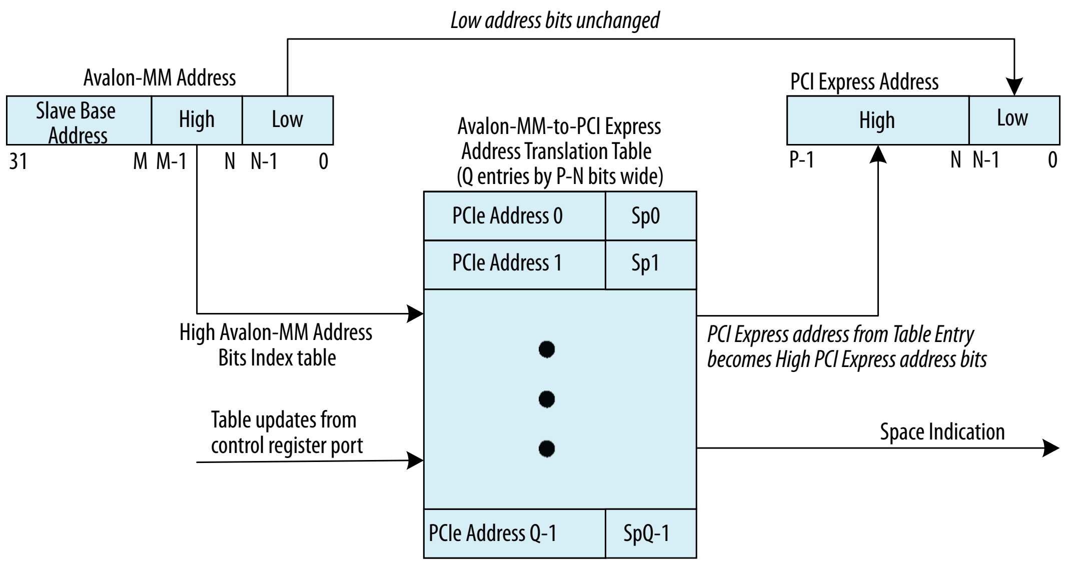
- Bits [31:24] select the TX slave module port from among other slaves connected to the same master by the system interconnect fabric. The decode is based on the base addresses assigned in Platform Designer.
- Bits [23:20] select the address translation table entry.
- Bits [63:20] of the address translation 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.

Figure 78. 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 parameters:

- *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.


Low address bits unchanged
Avalon-MM Address
Slave Base Address
High
Low
31
M
M-1
N
N-1
0
PCI Express Address
High
Low
P-1
N
N-1
0
Avalon-MM-to-PCI Express Address Translation Table (Q entries by P-N bits wide)
PCIe Address 0
Sp0
PCIe Address 1
Sp1
PCIe Address Q-1
SpQ-1
High Avalon-MM Address Bits Index table
Table updates from control register port
PCI Express address from Table Entry becomes High PCI Express address bits
Space Indication

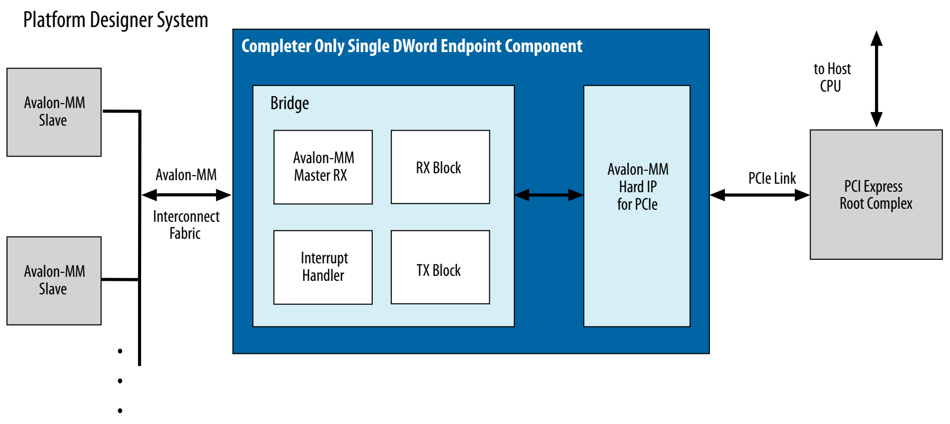
A.5. Completer Only Single Dword Endpoint

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 Platform Designer 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

Figure 79. Design Including Completer Only Single Dword Endpoint for PCI Express



The above figure shows the that completer-only single dword endpoint connects to a PCI Express Root Complex. A bridge component includes the Intel Arria 10 or Intel Cyclone 10 GX 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.

A.5.1. 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.

A.5.2. 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 *Avalon Memory Mapped Interfaces* in the Avalon Interface Specifications.

Related Information

- Avalon Interface Specifications
  For information about the Avalon-MM interface protocol.
- Avalon Interface Specifications

A.5.3. TX Block

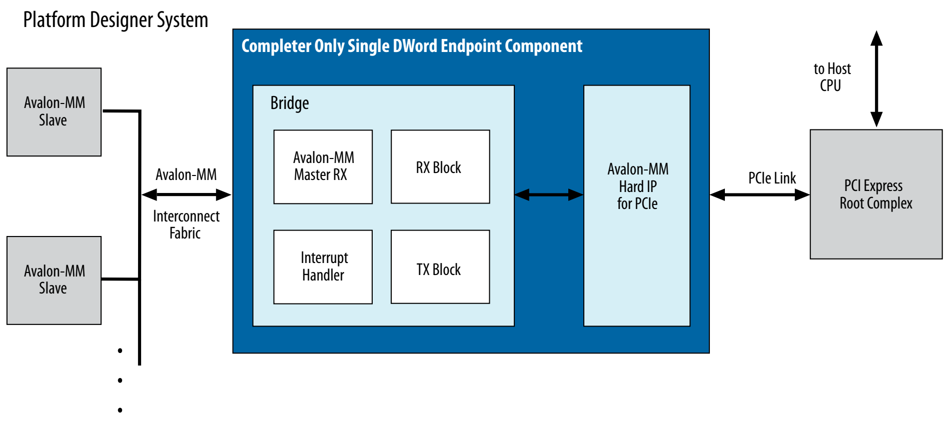
The TX block sends completion information to the Avalon-MM Hard IP for PCI Express which sends this information to the root complex. The TX completion block generates a completion packet with Completer Abort (CA) status and no completion data for unsupported requests. The TX completion block also supports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

A.5.4. 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_en able` bit is on, an MSI request is sent to the Intel Arria 10 or Intel Cyclone 10 GX 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 Platform Designer.

When the MSI registers in the Configuration Space of the Completer Only Single Dword Intel Arria 10 or Intel Cyclone 10 GX 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 80. Platform Designer 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.

A.5.5. Preliminary Support for Root Port

This release adds preliminary support for a Gen3 x4, Gen3 x8, and Gen2 x8 Root Port with a 256-bit Avalon-MM interface to the Application Layer.

This Avalon-MM Root Port Supports the following features:

- RX Master Module—This 256-bit bursting Avalon-MM master port propagates PCI Express requests, converting them to bursting read or write requests to the interconnect fabric.
- TX Slave Module—This optional 32-bit Avalon-MM slave port propagates single dword read and write requests from the interconnect fabric to the PCI Express link. The bridge translates requests from the interconnect fabric to PCI Express request packets.
- Control Register Access Slave Module—This optional, 32-bit Avalon-MM slave port provides access to internal control and status registers from external Avalon-MM masters. The CRA port supports single dword TLPS, including MemRd, MemWr, Message, and Configuration requests. It does not support bursting.

This preliminary release Gen3 Avalon-MM Root Port has the following limitations::

- It does not support legacy interrupts.
- The TX Slave Module does not support bursting.
- The TX Slave Module supports native PCI Express addresses. It does not translate Avalon-MM addresses to the PCI Express address space.

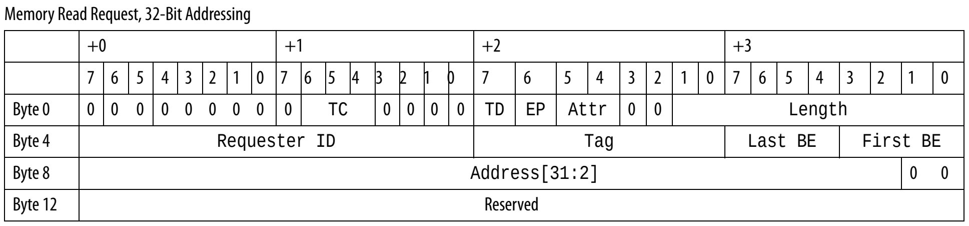
B. Transaction Layer Packet (TLP) Header Formats

The following sections show the TLP header formats for TLPs without a data payload, and for those with a data payload.

B.1. TLP Packet Formats without Data Payload

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

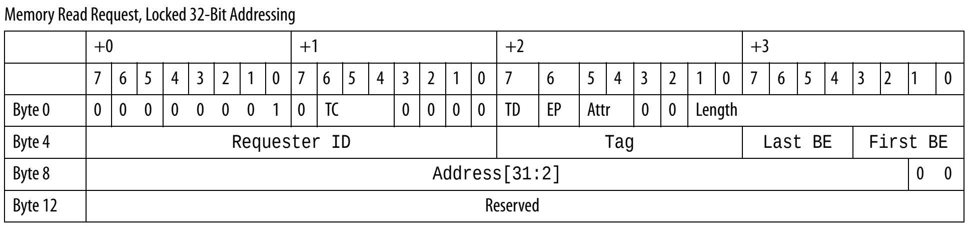
Figure 81. Memory Read Request, 32-Bit Addressing



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																															

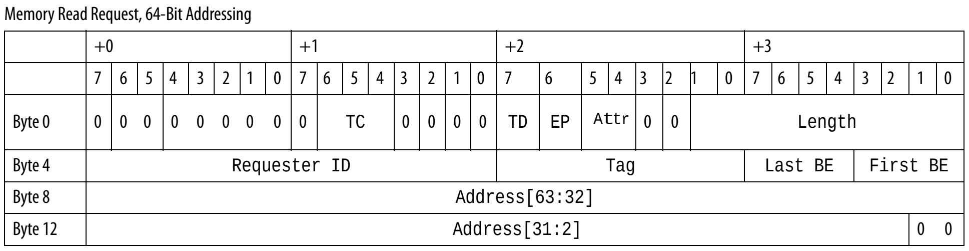
Figure 82. Memory Read Request, Locked 32-Bit Addressing



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 83. Memory Read Request, 64-Bit Addressing



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	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[63:32]																															
Byte 12	Address[31:2]																														0	0

Figure 84. Memory Read Request, Locked 64-Bit Addressing

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 85. Configuration Read Request Root Port (Type 1)

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 86. I/O Read Request

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 87. Message without Data

Message without Data

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

Note:

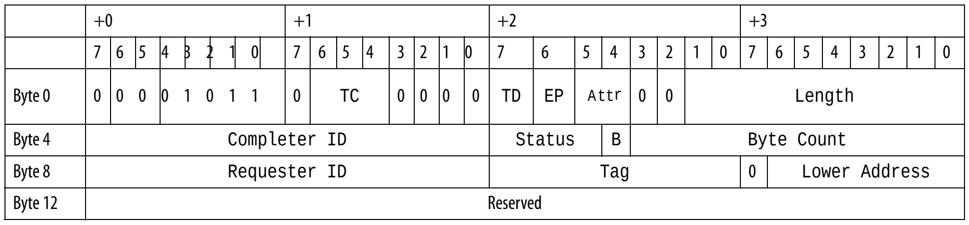
(1) Not supported in Avalon-MM.

Figure 88. Completion without Data

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 89. Completion Locked without Data

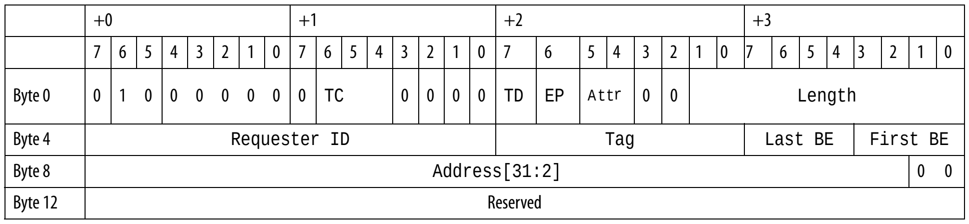


Completion Locked without Data

	+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																															

B.2. TLP Packet Formats with Data Payload

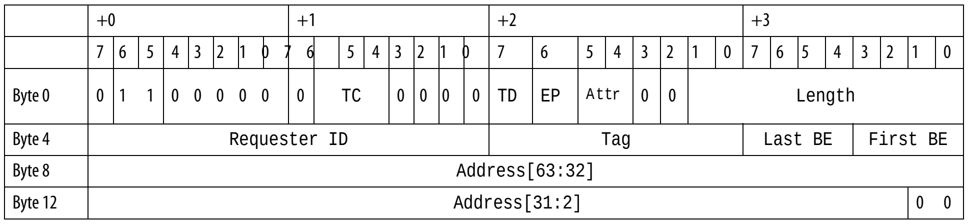
Figure 90. Memory Write Request, 32-Bit Addressing



Memory Write Request, 32-Bit Addressing

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

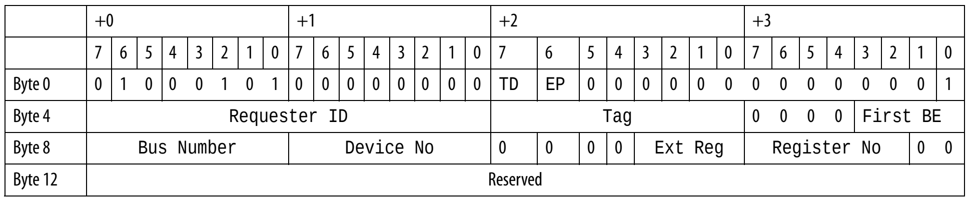
Figure 91. Memory Write Request, 64-Bit Addressing



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

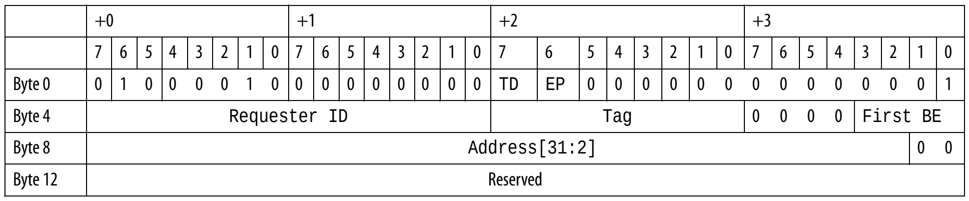
Figure 92. Configuration Write Request Root Port (Type 1)



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 93. I/O Write Request



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 94. Completion with Data

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 95. Completion Locked with Data

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 96. Message with Data

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																															

C. Lane Initialization and Reversal

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

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

Table 86. Lane Assignments without Lane Reversal

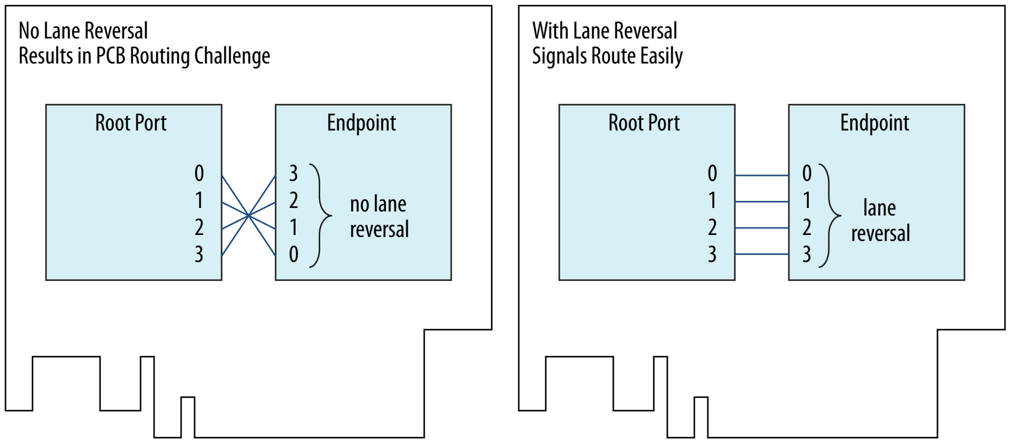
Lane Number	7	6	5	4	3	2	1	0
×8 IP core	7	6	5	4	3	2	1	0
×4 IP core	—	—	—	—	3	2	1	0
—	—	—	—	—	—	—	1	0
×1 IP core	—	—	—	—	—	—	—	0

Table 87. 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 pairings	7:0,6:1,5:2, 4:3, 3:4,2:5, 1:6,0:7	3:4,2:5, 1:6,0:7	1:6, 0:7	0:7	7:0,6:1, 5:2,4:3	3:0,2:1, 1:2,0:3	3:0, 2:1	3:0	7:0	3:0	1:0	0:0	

Figure 97. Using Lane Reversal to Solve PCB Routing Problems

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.



Send Feedback

D. Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM Interface for PCIe Solutions User Guide Archive

If an IP core version is not listed, the user guide for the previous IP core version applies.

IP Core Version	User Guide
17.0	Arria 10 Avalon-MM Interface for PCIe Solutions User Guide
16.1	Arria 10 Avalon-MM Interface for PCIe Solutions User Guide
16.0	Arria 10 Avalon-MM Interface for PCIe Solutions User Guide
15.1	Arria 10 Avalon-MM Interface for PCIe Solutions User Guide
15.0	Arria 10 Avalon-MM Interface for PCIe Solutions User Guide
14.1	Arria 10 Avalon-MM Interface for PCIe Solutions User Guide

E. Document Revision History

E.1. Document Revision History for the Intel Arria 10 Avalon-MM Interface for PCIe Solutions User Guide

Date	Version	Changes Made
2019.05.23	18.0	Added the note clarifying that the 24-bit Class Code register is divided into three 8-bit fields: Base Class Code, Sub-Class Code and Programming Interface.
2018.08.13	18.0	Added the step to invoke Vsim to the instructions for running ModelSim simulations.
2018.08.03	18.0	Made the following changes to the user guide: • Removed a reference to the internal descriptor controller since that is an Avalon-MM DMA feature and is covered in that User Guide. • Added text descriptions for the interfaces shown in Figure 27. • Updated signal names in Figure 27 to change rxm_bar0_* signals to rxm_bar<n>_* signals, where n is the BAR number and can range from 0 to 5. • Updated Tables 26 and 27 to change rxm_bar0_* signals to rxm_bar<n>_* signals. • Changed the title of Figure 30 to "Simultaneous RXM Read and RXM Write".
2018.05.07	18.0	Made the following changes to the user guide: • Changed Intel Cyclone 10 name to Intel Cyclone 10 GX. • Added CRA Read and Write timing diagrams. • Added missing link speeds/widths combinations to the Recommended Speed Grades table. • Removed any mention of static example designs in the Design Examples section. • Moved the topic on creating the .stp file from the *Design Implementation* chapter to the *Debugging* chapter. • Changed the title of the *Optional Features* chapter to *Additional Features* .
2017.10.06	17.1	Made the following changes to the user guide. • Added support for Intel Cyclone 10 GX devices. • Added **Enable RX-polarity inversion soft logic** parameter. • Corrected *Feature Comparison for all Hard IP for PCI Express IP Core* table: The Avalon-MM DMA interface does not automatically handle out-of-order completion s. • Rebranded as Intel.
2017.05.26	17.0	Made the following changes to the user guide: • Added note that starting with the Intel Quartus Prime Software, version 17.0, the QSF assignments in the following answer *What assignments do I need for a PCIe Gen1, Gen2 or Gen3 design that targets an Intel Arria 10 ES2, ES3 or production device?* are already included in the design.

continued...

Date	Version	Changes Made
2017.05.08	17.0	Made the following changes the IP core: • Added option soft DFE Controller IP on the **PHY** tab of the parameter editor to improve BER margin. The default for this option is off because it is typically not required. Short reflective links may benefit from this soft DFE controller IP. This parameter is available only for Gen3 configurations. Made the following changes to the user guide: • Updated *PCI Express Gen3 Bank Usage Restrictions* status. These restrictions affect all Arria 10 ES and production devices. • Corrected *Feature Comparison for all Hard IP for PCI Express IP Cores* table. Out-of-order Completions are not supported transparently for the Avalon-MM with DMA interface. • Added Gen3 x8 Root Port to the *Recommended Speed Grades for All Avalon-MM Widths and Frequencies* table. • Corrected default values for the *Uncorrectable Internal Error Mask Register* and *Correctable Internal Error Mask Register* registers. • Corrected minor errors and typos.
2017.03.15	17.0	• Removed Gen3x8 256-bit interface from *Recommended Speed Grades* table. This configuration is not supported for the Avalon-MM interface. • Added statement that Intel Arria 10 devices do not support the **Create timing and resource estimates for third-party EDA synthesis tools** option on the **Generate ➤ Generate HDL** menu. • Rebranded as Intel.
2016.10.31	16.1	Made the following changes to the IP core: • Changed timing models support to final for most Intel Arria 10 device packages. Exceptions include some military and automotive speed grades with extended temperature ranges. • Added parameter to select the requested preset for Phase2 and Phase3 far-end TX equalization. Made the following changes to the user guide: • Corrected the number of tags supported in the *Feature Comparison for all Hard IP for PCI Express IP Cores* table. • Removed recommendations about connecting `pin_perst`. These recommendations do not apply to Arria 10 devices. • Added PCIe bifurcation to the *Feature Comparison for all Hard IP for PCI Express IP Cores* table. PCI bifurcation is not supported. • Changed the recommended value of `test_in[31:0]` from 0xa8 to 0x188. • Removed *Configuration Space Register Access Timing* timing diagram. These signals are not available at the top-level for the Avalon-MM interface. • Added instructions for turning on autonomous mode in the Quartus Prime software. • Added -3 to recommended speed grades for the 125 MHz interface.
2016.05.20	16.0	Added preliminary support for a Gen3 x8 Root Port using a 256-bit interface to the Application Layer. Added support for Intel FPGA IP Evaluation Mode in the Quartus Prime Pro Edition software. Added automatic generation of basic Signal Tap Logic Analyzer files to facilitate debugging. The PIO Design Examples included in the *Quick Start Guide* now support 64- and 128-bit interfaces to the Application Layer. (The 15.1 release supported only a 256-bit interface to the Application Layer interface.) Updated figures in *Physical Layout of Hard IP in Intel Arria 10 Devices* to include more detail about transceiver banks and channel restrictions. Revised description of `TxsByteEnable_i[<w>-1:0]` signal. This signal qualifies both read and write data. Added Gen3 x2 128-bit interface with 125 MHz clock to the `coreclkout_hip` *Application Layer Clock Frequency for All Combinations of Link Width, Data Rate and Application Layer Interface Widths* table.

continued...

Date	Version	Changes Made
		Clarified optimal read request size for typical systems that include the Avalon-MM TX slave interface. In the *Getting Started with the Hard IP for PCI Express with the Avalon-MM Interface* chapter, changed the instructions to use specify the 10AX115S2F45I1SG device which is used on the Intel Arria 10 GX FPGA Development Kit - Production (not ES2) Edition. Added **Vendor Specific Extended Capability (VSEC) Revision** and **User Device or Board Type ID register from the Vendor Specific Extended Capability** to the **VSEC** tab of the component GUI. Added statement that the testbench can only simulate a single Endpoint or Root Port at a time. Enhanced statements covering the deficiencies of the Altera-provided testbench. Updated signal names to match those shown in the figure *64- or 128-Bit Avalon-MM Interface to the Application Layer*. Added transceiver bank usage placement restrictions for Gen3 ES3 devices. Removed support for -3 speed grade devices. Corrected minor errors and typos.
2015.11.02	15.1	Added new **Generate Design Example** option that automatically generates both simulation and hardware design examples with the parameters you specify. You can download the hardware design example directly to the Intel Arria 10 GX FPGA Development Kit. Added preliminary support for Gen3 x4, Gen3 x8, and Gen2 x8 Root Port using a 256-bit Avalon-MM interface to the Application Layer. Improved component GUI that simplifies parameterization. Among the changes is a new single parameter, **HIP mode** that combines all supported data rates, interface widths and frequencies as a single parameter. Added support for Completion buffer overflow monitoring. Improved the definition of `npor`. Removed **Legacy Endpoint** option for **Port Type** parameter. Clarified Application Layer requirements for multiple and single MSI and MSI-X support. Corrected width of AVL_IRQ. It is 16 bits. Added clarification for the use of byte enables with the 128-bit Avalon-MM bridge. Supported patterns for byte enables must be at the dword granularity. Clarified Avalon-MM addressing for various data widths. Renamed the data rate `sim_pipe_rate` to `rate` to match renaming for Intel Arria 10 devices.Made the following changes: Removed support for the RP_RXCPL_STATUS field of the Root Port request register at 0x2010. Added signal descriptions for optional hard IP status bus. Fixed minor errors and typos.
2015.06.05	15.0	Added note in i*Physical Layout of Hard IP in Arria 10 Devices* to explain Intel Arria 10 design constraint that requires that if the lower HIP on one side of the device is configured with a Gen3 x4 or Gen3 x8 IP core, and the upper HIP on the same side of the device is also configured with a Gen3 IP core, then the upper HIP must be configured with a x4 or x8 IP core.
2015.05.14	15.0	Made the following changes to the user guide: • Added **Enable Hard IP Status Bus when using the AVMM interface** parameter in *Interface System Settings*. This parameter is available in the IP core v15.0 and later.
2015.05.04	15.0	Made the following changes to the user guide:

continued...

Date	Version	Changes Made
		• Enhanced the descriptions in i*Avalon-MM-to-PCI Express Address Translation Table*. • Added **Enable Altera Debug Master Endpoint (ADME)** parameter to support optional Native PHY register programming with the Altera System Console. • Added support to send message TLPs with data payload of any length from a Root Port. Refer to *Programming Model for Avalon-MM Root Port* and to the new supported TLP entry for Avalon-MM variations in the Feature Comparison for all Hard IP for PCI Express IP Cores table in *Features*. • Added information about the new custom design examples, in the *Design Examples* section. • Added column for Avalon-ST Interface with SR-IOV variations in Feature Comparison for all Hard IP for PCI Express IP Cores table in the *Features* section. • Enhanced descriptions of channel placement, added fPLL placement for Gen1 and Gen2 data rates, and added master CGB location, in *Physical Layout of Hard IP in Arria 10 Devices*. • Updated DUT module name in testbench and design example figures. • Removed list of static design examples from Design Examples on page 12. You can derive the list from the installation directory where design examples are available. • Removed Migration and TLP Format appendices, and added new *Frequently Asked Questions* appendix. • Reorganized sections in Debugging on page 154 and Setting Up Simulation on page 147. Removed *Reducing Counter Values for Serial Simulations* section, which is no longer relevant. Default counter values are automatically set for simulation. • Updated information in *SDC Timing Constraints*. • Fixed minor errors and typos.
2014.12.15	14.1	Made the following changes: • Revised Root Port programming model description, *Receiving a Completion TLP*, to cover read and non-posted completions. • Added *Avalon-MM Testbench and Design Example* chapter. • Added statement that the bottom left hard IP block includes the CvP functionality for flip chip packages. For other package types, the CvP functionality is in the bottom right block. • Corrected bit definitions for CvP Status register. • Updated definition of `CVP_NUMCLKS` in the CvP Mode Control register. • Added definitions for `test_in[2]`, `test_in[6]`, and `test_in[7]`. • Revised discussion of SDC files to include in Quartus II project.
2014.08.18	14.0 Intel Arria 10	Made the following changes to the Intel Arria 10 Avalon-MM Hard IP for PCI Express • Optionally changed the `cra_address` to 14 bits from 12. • Added simulation log file, `altpcie_monitor_a10_dlhip_tlp_file_log.log`, that is automatically generated in your simulation directory. To simulate in the Quartus II 14.0 software release, you must regenerate your IP core to create the supporting monitor file the generates `altpcie_monitor_a10_dlhip_tlp_file_log.log`. Refer to *Understanding Simulation Dump File Generation* for details. • Added support for 64-bit addressing, making address translation unnecessary. • Removed *Channel Placement for PCIe in Arria 10 Devices*. Please contact your Altera sales representative for PLL and channel usage.

continued...

Date	Version	Changes Made
		• Added simulation support for Phase 2 and Phase 3 equalization when requested by third-party BFM. • Added restrictions on the legal patterns of enabled and disabled bytes for `txs_byteenable[<w>-1:0]`. • Changed the PIPE interface to 32 bits for all data rates. This change requires you to recompile your 13.1 variant in 14.0. Made the following changes to the user guide: • Changed device part number for *Getting Started* chapter to **10AX115R2F40I2LG**. • Corrected frequency range for `hip_reconfig_clk`. It should be 100-125 MHz. • Clarified the behavior of the `txs_waitrequest` signal. • Added statement that the bottom left hard IP block includes the CvP functionality for flip chip packages. For other package types, the CvP functionality is in the bottom right block. • Simplified the *Getting Started* chapter. It copies the example from the install directory and does not include step-by-step instructions to recreate the design. • Removed 125 MHz clock as optional `refclk` frequency in Intel Arria 10 devices. Intel Arria 10 devices support an 100 MHz reference clock as specified by the *PCI Express Base Specification, Rev 3.0*. • Added definitions for `test_in[2]`, `test_in[6]` and `test_in[7]`. • Clarified that the Avalon-MM Bridge does not generate out-of-order Avalon-MM-to-PCI Express Read Completions even to different BARs. • Added sections on making analog QSF and pin assignments. • Enhanced the definition of Device ID and Sub-system Vendor ID to say that these registers are only valid in the Type 0 (Endpoint) Configuration Space. • Updated *Power Supply Voltage Requirements* table. • Removed all references to the Avalon-MM interrupt vector register. This register is not used. • Corrected values for **Maximum payload size** parameter. The sizes available are 128 or 256 bytes. • Removed `txdatavalid0` signal from the PIPE interface. This signal is not available. • Updated *Power Supply Voltage Requirements* table. • Updated *Physical Placement of the Arria 10 Hard IP for PCIe IP and Channels* to show GT devices instead of GX devices. • Corrected bit definitions for `CvP Status` register. • Updated definition of CVP_NUMCLKS in the `CvP Mode Control` register. • Removed discussion of `pclk`. This clock is not customer accessible in Intel Arria 10 devices. • Removed PLL from channel placement figures. • Added fast passive parallel (FPP) to supported configuration schemes in *CvP in Intel Arria 10 Devices* figure. • Corrected *Reset Controller in Intel Arria 10 Devices* figure in *Reset and Clocks* chapter. • Corrected bit definitions for `CvP Status` register.
2013.12.02	13.1 Intel Arria 10	Initial release.