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

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

Subscribe
Send Feedback

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

Contents

1. Datasheet

1.1. Intel® Arria® 10 or Intel® Cyclone® 10 GX Avalon-MM DMA 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 the *PCI Express Base Specification 3.0* and *PCI Express Base Specification 2.0* respectively.

The Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express with the Avalon® Memory-Mapped (Avalon-MM) DMA interface removes some of the complexities associated with the PCIe protocol. For example, the IP core handles TLP encoding and decoding. In addition, the IP core includes Read DMA and Write DMA engines. If you have already architected your own DMA system with the Avalon-MM interface, you may want to continue to use that system. However, you may benefit from the simplicity of having the included DMA engines. This variant is available in Platform Designer for 128- and 256-bit interfaces to the Application Layer. The Avalon-MM interface and DMA engines are implemented in FPGA soft logic.

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

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

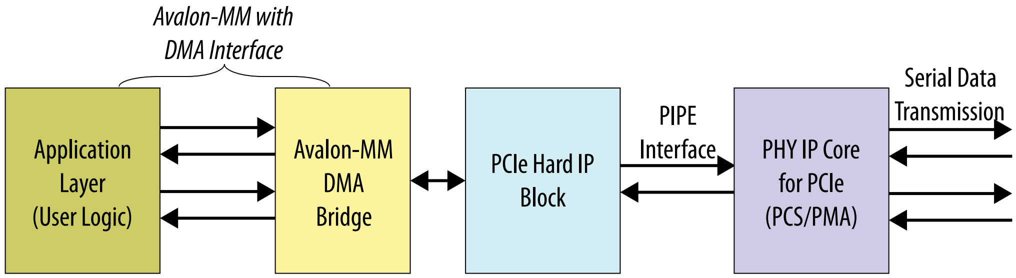


Table 1. Intel Arria 10 PCI Express Maximum Theoretical Data Throughput

The following table shows the aggregate bandwidth of a PCI Express link for Gen1, Gen2, and Gen3 for 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 introduces only a 1.5% overhead.

Note: Maximum theoretical data throughput values are link throughput values after accounting for coding overhead. The real application throughput is reduced by several other factors. Refer to the *Understanding Throughput* section of AN 456 for more details.

Units are Gigabits per second (Gbps).

	Link Width		
	×2	**×4**	**×8**
PCI Express Gen1 (2.5 Gbps)	Not supported	Not supported	16 Gbps
PCI Express Gen2 (5.0 Gbps)	Not supported	16 Gbps	32 Gbps
PCI Express Gen3 (8.0 Gbps)	15.75 Gbps	31.51 Gbps	63Gbps

Table 2. Intel Cyclone 10 GX PCI Express Maximum Theoretical Data Throughput

The following table shows the aggregate bandwidth of a PCI Express link for Gen1and Gen2 for 4 lanes. The protocol specifies 2.5 giga-transfers per second for Gen1 and 5.0 giga-transfers per second for Gen2. 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.

Units are Gigabits per second (Gbps).

	Link Width	
	×2	**×4**
PCI Express Gen1 (2.5 Gbps)	Not supported	Not supported
PCI Express Gen2 (5.0 Gbps)	Not supported	16 Gbps

Related Information

- Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA Interface for PCIe Solutions User Guide Archive on page 126
- 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.
- AN 690: PCI Express Avalon-MM DMA Reference Design
  For a reference design that illustrates chaining DMA performance using internal memory.
- 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 Avalon-MM DMA for PCI Express supports the following features:

- Complete protocol stack including the Transaction, Data Link, and Physical Layers implemented as hard IP.
- Native support for Gen1 x8, Gen2 x4, Gen2 x8, Gen3 x2, Gen3 x4, Gen3 x8 for Endpoints for Intel Arria 10 devices. The variant downtrains when plugged into a lesser link width or a system which supports a lower maximum link rate.
- Native support for Gen2 x4 for Endpoints for Intel Cyclone 10 GX devices. The variant downtrains when plugged into a lesser link width or a system which supports a lower maximum link rate.
- Dedicated 16 KB receive buffer.
- Support for 128- or 256-bit Avalon-MM interface to Application Layer with embedded DMA up to Gen3 ×8 data rate for Intel Arria 10.
- Support for 128-bit Avalon-MM interface to Application Layer with embedded DMA for Gen2 x4 for Intel Cyclone 10 GX devices.
- 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 3. Comparison for 128- and 256-Bit Avalon-MM with DMA Interface to the Application Layer

Intel Cyclone 10 GX devices only support the Gen2 x4 configuration.

Feature	128-Bit Interface	256-Bit Interface [(1)]
Gen1	x8	Not supported
Gen2	x4, x8	x8
Gen3 [(2)]	x2, x4	x4, x8

continued...

(1) Intel Cyclone 10 GX devices only support up to the Gen2 x4 configuration.

Feature	128-Bit Interface	256-Bit Interface [1]
Root Port	Supported	Supported
Tags supported	16	16 or 256
Maximum descriptor size	1 MB	1 MB
Maximum payload size	128 or 256	128 or 256
Immediate write[3]	Not supported	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.

1.3. Comparison of Avalon-ST, Avalon-MM and Avalon-MM with DMA Interfaces

Table 4. 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 [4]	×2, ×4, ×8 [5]
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	16 or 256

continued...

(1) Intel Cyclone 10 GX devices only support up to the Gen2 x4 configuration.

(2) Intel Cyclone 10 GX devices do not support Gen3 configurations.

(3) The `Immediate Write` provides a fast mechanism to send a Write TLP upstream. The descriptor stores the 32-bit payload, replacing the `Source Low Address` field of the descriptor.

(4) Gen3 x8 Avalon-MM is supported in Root Port mode only

(5) Gen3 x8 Avalon-MM DMA is supported in Endpoint mode only

Send Feedback

Feature	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA
Automatically handle out-of-order completions (transparent to the Application Layer)	Not supported	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	1, 2, 4, 8, 16, or 32
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: This table applies across device families. For specific widths and speeds supported by the Intel Arria 10 or Intel Cyclone 10 GX devices, refer to the tables in the *Features* section.

Table 5. 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/RP (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/RP (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

continued...

TLP (Transmit Support)	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA
Message Request with Data (`MsgD`)	EP/RP	Not supported	Not supported
Completion (`Cpl`)	EP/RP	EP/RP	EP/RP (Read & Write DMA Avalon-MM Masters)
Completion with Data (`CplD`)	EP/RP	Not supported	EP/RP (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 DMA 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*.

Related Information

- Intel Arria 10 and Intel Cyclone 10 GX Avalon-MM Interface for PCIe Solutions User Guide
  For the Avalon-MM interface with no DMA.
- Intel Arria 10 and Intel Cyclone 10 GX Avalon-ST Interface for PCIe Solutions User Guide
  For the Avalon-ST interface.
- Intel 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).
- PCI Express Base Specification 3.0

1.4. Release Information

Table 6. 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	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.

1.5. 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 7. 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

1.6. Design Examples

Platform Designer example designs are available for the Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA 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.

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

Related Information

Getting Started with the Avalon-MM DMA on page 15

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 with DMA Intel Arria 10 or Intel Cyclone 10 GX variants include an Avalon-MM DMA bridge implemented in soft logic that operates as a front end to the hardened protocol stack. The following table shows the typical expected device resource utilization for selected configurations using the current version of the Quartus Prime software targeting an Intel Arria 10 or Intel Cyclone 10 GX device. With the exception of M20K memory blocks, the numbers of ALMs and logic registers are rounded up to the nearest 50.

Table 8. Resource Utilization Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA for PCI Express

Data Rate, Number of Lanes, and Interface Width	ALMs	M20K Memory Blocks	Logic Registers
Gen2 x4 128	4300	29	5800
Gen2 x8 128	12700	19	22300
Gen3 x8 256	18000	47	31450

Related Information

Running the Fitter

1.9. Recommended Speed Grades

Table 9. Intel Cyclone 10 GX Recommended Speed Grades for All Avalon-MM with DMA Widths and Frequencies

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

Lane Rate	Link Width	Interface Width	Application Clock Frequency (MHz)	Recommended Speed Grades
Gen2	×4	128 bits	125	–5, –6

Table 10. Intel Arria 10 Recommended Speed Grades for All Avalon-MM with DMA Widths and Frequencies

Lane Rate	Link Width	Interface Width	Application Clock Frequency (MHz)	Recommended Speed Grades
Gen1	×8	128 Bits	125	–1, –2, –3
Gen2	×4	128 bits	125	–1, –2, –3
	×8	128 bits	250	–1, –2
Gen3	×2	128 bits	125	–1, –2, –3
	×4	128 bits	250	–1, –2
	×8	256 bits	250	–1, –2

Related Information

- 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 22
- Getting Started with the Avalon-MM DMA on page 15
- 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. Getting Started with the Avalon-MM DMA

You can download this Platform Designer design example, `ep_g3x8_avmm256_integrated.qsys`, from the `<install_dir>/ ip/altera/altera_pcie/altera_pcie_a10_ed/example_design/a10` directory.

The design example includes the following components:

Avalon-MM DMA for PCI Express

This IP core includes highly efficient DMA Read and DMA Write modules. The DMA Read and Write modules effectively move large blocks of data between the PCI Express address domain and the Avalon-MM address domain using burst data transfers. Depending on the configuration you select, the DMA Read and DMA Write modules use either a 128- or 256-bit Avalon-MM datapath.

In addition to high performance data transfer, the DMA Read and DMA Write modules ensure that the requests on the PCI link adhere to the *PCI Express Base Specification, 3.0*. The DMA Read and Write engines also perform the following functions:

- Divide the original request into multiple requests to avoid crossing 4KByte boundaries.
- Divide the original request into multiple requests to ensure that the maximum payload size is equal to or smaller than the maximum payload size for write requests and maximum read request size for read requests.
- Supports out-of-order completions when the original request is divided into multiple requests to adhere to the read request size. The Read Completions can come back in any order. The Read DMA Avalon-MM master port supports out-of-order Completions by writing the Read Completions to the correct locations. The Read DMA Avalon-MM master port does not have an internal reordering buffer.

Using the DMA Read and DMA Write modules, you can specify descriptor entry table entries with large payloads.

On-Chip Memory IP core

This IP core stores the DMA data. This memory has a 256-bit data width.

Descriptor Controller

The Descriptor Controller manages the Read DMA and Write DMA modules. Host software programs the Descriptor Controller internal registers with the location of the descriptor table. The Descriptor Controller instructs the Read DMA module to copy the entire table to its internal FIFO. It then pushes the table entries to DMA Read or DMA Write modules to transfer data. The Descriptor Controller also sends DMA status upstream via an Avalon-MM TX slave port.

In this example design the Descriptor Controller parameter, **Instantiate internal descriptor controller**, is on. Consequently, the Descriptor Controller is integrated into the Avalon-MM DMA bridge as shown in the figure below. Embedding the

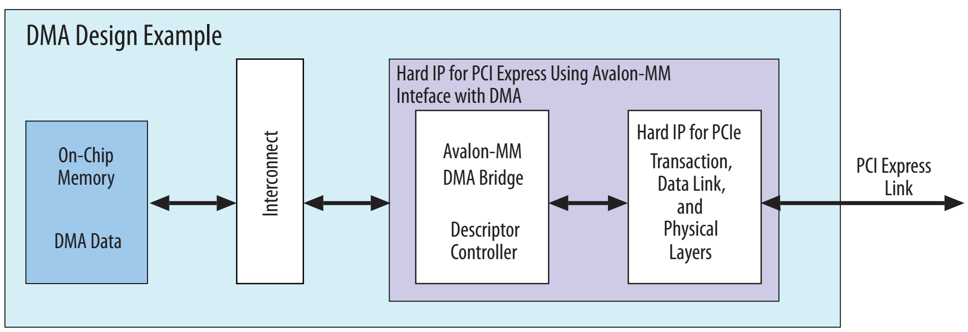
Descriptor Controller in the Avalon-MM DMA bridge simplifies the design. If you plan to replace the Descriptor Controller IP core with your own implementation, do not turn on the **Instantiate internal descriptor controller** in the parameter editor when parameterizing the IP core.

The Descriptor Controller supports the following features:

- A single duplex channel.
- Minimum transfer size of one dword (4 bytes).
- Maximum transfer size of 1 M (1024 * 1024) - 4 bytes.

  Note: Although the Descriptor Controller supports a maximum transfer size of (1 M (1024 * 1024) - 4 bytes), the on-chip memory in this design example is smaller. Consequently, this design example cannot handle the maximum transfer size.
- Endpoints, only.
- Provides status to host software by generating an MSI interrupt when the DMA transfer completes.

Figure 2. Block Diagram of the Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA for PCI Express



Design Example Limitations

This design example is intended to show basic DMA functionality. It is not a substitute for a robust verification testbench. If you modify this testbench, be sure to verify that the modifications result in the correct behavior.

Related Information

- Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA for PCI Express on page 114
- DMA Descriptor Controller Registers on page 80

2.1. Understanding the Avalon-MM DMA Ports

The Avalon-MM DMA bridge includes ports to implement the DMA functionality. The following figure and table below illustrate and describe these ports.

Figure 3. Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA for PCI Express Platform Designer System Design

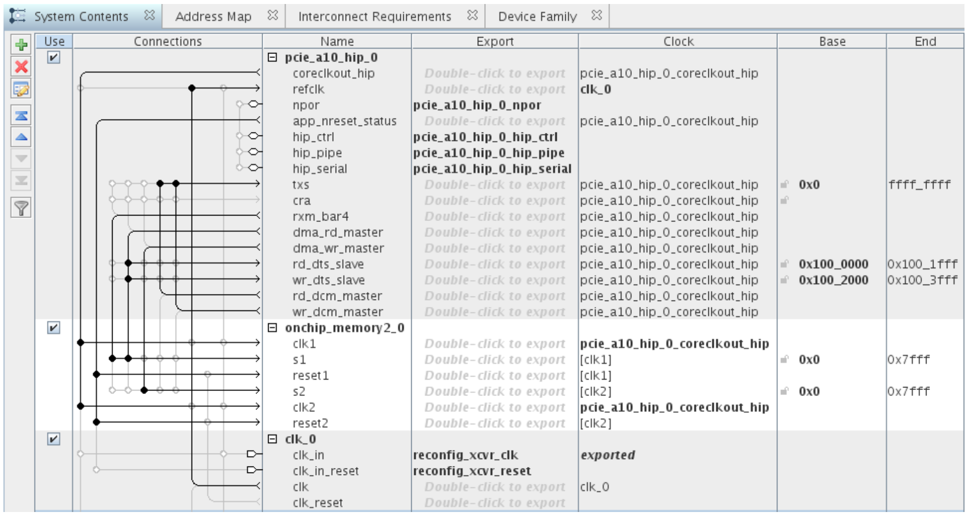


Table 11. Avalon-MM DMA Platform Designer System Port Descriptions

Function	Port	Description
TXS	`Txs`	This is an Avalon-MM slave port. In a typical application, an Avalon-MM master uses this port to send memory reads or writes to the PCIe domain. The Descriptor Controller uses it to write DMA status back to descriptor space in the PCIe domain when the DMA completes its operation. The Descriptor Controller also uses this port to send MSI interrupts upstream.
Read Data Mover	`dma_rd_master`	This is an Avalon-MM master port. The Read Data Mover moves data from the PCIe domain to the on-chip memory during normal read DMA operation. The Read Data Mover also fetches the descriptors from the PCIe domain and writes them to the FIFO in the Descriptor Controller. There are two separate descriptor tables for the read and write write DMA descriptors.The `dma_rd_master` connects to `wr_dts_slave` port to load the write DMA descriptor FIFO and `rd_dts_slave` port to load the read DMA descriptor FIFO.
Write Data Mover	`dma_wr_master`	This is an Avalon-MM master port. The Write Data Mover reads data from the on-chip memory and then writes data to the PCIe domain.
Descriptor Controller FIFOs	`wr_dts_slave` and `rd_dts_slave`	This is an Avalon-MM slave port for the Descriptor Controller FIFOs. When the Read Data Mover fetches the descriptors from system memory, it writes the descriptors to the FIFO using this port. Because there are separate descriptor tables for read and write, there are two ports. The address range for the write DMA FIFO is `0x100_0000`—`0x100_1FFF`. The address range for the read DMA FIFO is `0x100_2000`—`0x100_3FFF`.
Control in the Descriptor Controller	`wr_dcm_master` and `rd_dcm_master`	The control block in the Descriptor Controller has one transmit and one receive port, one for read DMA and another one for write DMA. The receive port connects to the RXM_BAR0 and the transmit port connects to the `Txs`.

continued...

Function	Port	Description
		The receive path from the RXM_BAR0 connects internally. It is not shown in the connections panel. For the transmit path, both read and write DMA ports connect to the Txs externally as shown in the connections panel.
RXM_BAR0	not shown in connections panel	This is an Avalon-MM master port. It passes the memory access from PCIe host to PCIe BAR0. The host uses this port to program the Descriptor Controller. Because this Platform Designer system uses an internal descriptor controller, the port connection is not shown in Platform Designer. The connection is inside the a10_pcie_hip_0 module.
RXM_BAR4	Rxm_BAR4	This is an Avalon-MM master port. It passes the memory access from PCIe host to PCIe BAR4. In the Platform Designer system, it connects to the on-chip memory. The PCIe host accesses the memory through PCIe BAR4 In a typical application, system software controls this port to initialize random data in the on-chip memory. Software also reads the data back to verify correct operation.

2.2. Generating the Testbench

1. Copy the example design, `ep_g3x8_avmm256_integrated.qsys`, from the installation directory: `<install_dir>/ip/altera/altera_pcie/altera_pcie_a10_ed/example_design/a10/` to your working directory.
2. Start Platform Designer, by typing the following command:

   `qsys-edit`
3. Open `ep_g3x8_avmm256_integrated.qsys`.
4. Click **Generate ➤ Generate Testbench System**.
5. Specify the following parameters:

Table 12. Parameters to Specify in the Generation Dialog Box

Parameter	Value
Testbench System	
Create testbench Platform Designer system	**Standard, BFMs for standard Platform Designer interfaces**
Create testbench simulation model	**Verilog**
Allow mixed-language simulation	You can leave this option off.
Output Directory	
Testbench	`<working_dir>/ep_g3x8_avmm256_integrated_tb`

6. Click **Generate**.

   Note: Intel Arria 10 or Intel Cyclone 10 GX devices do not support the **Create timing and resource estimates for third-party EDA synthesis tools** option on the **Generate ➤ Generate HDL** menu. You can select this menu item, but generation fails.

2.2.1. Understanding the Simulation Generated Files

Table 13. Platform Designer Generation Output Files

Directory	Description
`<working_dir>/ ep_g3x8_avmm256_integrated_tb/ ep_g3x8_avmm256_integrated_tb/`	Includes directories for all components of the testbench. Also includes the following files: • Simulation Package Descriptor File (`.spd`) which lists the required simulation files • Comma-Separated Value File (`.csv`) describing the files in the testbench
`<working_dir>/ ep_g3x8_avmm256_integrated_tb/ ep_g3x8_avmm256_integrated_tb/sim/ <cad_vendor>/`	Includes testbench subdirectories for the Aldec, Cadence, Mentor, and Synopsys simulation tools with the required libraries and simulation scripts.

2.2.2. Understanding Simulation Log File Generation

Starting with the Quartus II 14.0 software release, simulation automatically creates a log file, `altpcie_monitor_<dev>_dlhip_tlp_file_log.log` in your simulation directory.

Table 14. Sample Simulation Log File Entries

Time	TLP Type	Payload (Bytes)	TLP Header
17989 RX	CfgRd0	0004	04000001_0000000F_01080008
17989 RX	MRd	0000	00000000_00000000_01080000
18021 RX	CfgRd0	0004	04000001_0000010F_0108002C
18053 RX	CfgRd0	0004	04000001_0000030F_0108003C
18085 RX	MRd	0000	00000000_00000000_0108000C

2.3. Simulating the Example Design in ModelSim

1. In a terminal, change directory to `<workingdir>/pcie_g3x8_integrated_tb/ep_g3x8_avmm256_integrated_tb/sim/mentor`.
2. Start the ModelSim simulator (invoke vsim).
3. To run the simulation, type the following commands in a terminal window:
   a. `do msim_setup.tcl`
   b. `ld_debug`
      The `ld_debug` command compiles all design files and elaborates the top-level design without any optimization.
   c. `run -all`

The simulation performs the following operations:

- Various configuration accesses after the link is initialized
- Setup of the DMA controller to read data from the BFM's shared memory
- Setup of the DMA controller to write the same data back to the BFM's shared memory
- Data comparison and report of any mismatch

2.4. Running a Gate-Level Simulation

The PCI Express testbenches run simulations at the register transfer level (RTL). However, it is possible to create your own gate-level simulations. Contact your Intel Sales Representative for instructions and an example that illustrates how to create a gate-level simulation from the RTL testbench.

2.5. Generating Synthesis Files

1. On the **Generate** menu, select **Generate HDL**.
2. For **Create HDL design files for synthesis**, select **Verilog**.

   You can leave the default settings for all other items.
3. Click **Generate** to generate files for synthesis.
4. Click **Finish** when the generation completes.

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.

2.6. Compiling the Design

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

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

2.7. Creating a Quartus Prime Project

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

1. On the Quartus Prime File menu, click then **New Project Wizard**, then **Next**.
2. Click **Next** in the **New Project Wizard: Introduction** (The introduction does not appear if you previously turned it off.)
3. On the **Directory, Name, Top-Level Entity** page, enter the following information:

   a. For **What is the working directory for this project**, browse to `<project_dir>/ep_g3x8_avmm256_integrated/`.
   b. For **What is the name of this project?** browse to the `<project_dir>/ep_g3x8_avmm256_integrated/synth` directory and select `ep_g3x8_avmm256_integrated.v`.
   c. Click **Next**.
4. For **Project Type** select **Empty project**.
5. Click **Next**.
6. On the **Add Files** page, add `<project_dir>/ep_g3x8_avmm256_integrated/synth/ep_g3x8_avmm256_integrated.qip` to your Quartus Prime project.Click
7. Click **Next** to display the **Family & Device Settings** page.
8. On the **Device** page, choose the following target device family and options:
   a. In the **Family** list, select **Intel Arria 10 (GX/SX/GT) or Intel Cyclone 10 GX**.
   b. In the **Devices** list, select **All**.
   c. In the **Available devices** list, select the appropriate device. For Intel Arria 10 FPGA Development Kit, select **10AX115S2F45I1SG**.

   Note: Currently, you cannot target an Intel Cyclone 10 GX Development Kit when generating an example design for Intel Cyclone 10 GX.
9. Click **Next** to close this page and display the **EDA Tool Settings** page.
10. From the **Simulation** list, select **ModelSim**. From the **Format** list, select the HDL language you intend to use for simulation.
11. Click **Next** to display the **Summary** page.
12. Check the **Summary** page to ensure that you have entered all the information correctly.
13. Click **Finish**.
14. Save your project.

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 15. 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 16. 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.

3.1.1. RX Buffer Allocation Selections Available by Interface Type

Table 17. RX Buffer Allocation Selections Available by Interface Type

Interface Type	Minimum	Low	Balanced	High	Maximum
Avalon-ST	Available	Available	Available	Available	Available
Avalon-MM	Available	Available	Available	Not Available	Not Available
Avalon-MM with DMA	Available	Available	Available	Not Available	Not Available
Avalon-ST with SR-IOV	Available	Available	Available	Available	Available

3.2. Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM Settings

Table 18. Avalon Memory-Mapped System Settings

Parameter	Value	Description
Avalon-MM address width	**32-bit** **64-bit**	Specifies the address width for Avalon-MM RX master ports that access Avalon-MM slaves in the Avalon address domain. When you select 32-bit addresses, the PCI Express Avalon-MM DMA bridge performs address translation. When you specify 64-bits addresses, no address translation is performed in either direction. The destination address specified is forwarded to the Avalon-MM interface without any changes. For the Avalon-MM interface with DMA, this value must be set to **64**.
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 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
Instantiate Internal Descriptor Controller	**On/Off**	When you turn this option **On**, the descriptor controller is included in the Avalon-MM DMA bridge. When you turn this option off, the descriptor controller should be included as a separate external component. Turn this option on, if you plan to use the Intel-provided descriptor controller in your design. Turn this option off if you plan to modify or replace the descriptor controller logic in your design.
Enable burst capabilities for RXM BAR2 port	**On/Off**	When you turn this option **On**, the BAR2 RX Avalon-MM masters is burst capable. If BAR2 is 32 bits and Burst capable, then BAR3 is not available for other use. If BAR2 is 64 bits, the BAR3 register holds the upper 32 bits of the address.
Enable 256 tags	**On/Off**	When you turn this option **On**, the core supports 256 tags, improving the performance of high latency systems. Turning this option on turns on the `Extended Tag` bit in the `Control` register.
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

The type and size of BARs available depend on port type.

Table 19. BAR Registers

Parameter	Value	Description
Type	**Disabled** **64-bit prefetchable memory** **32-bit non-prefetchable memory** **32-bit prefetchable memory** **I/O address space**	If you select 64-bit prefetchable memory, 2 contiguous BARs are combined to form a 64-bit prefetchable BAR; you must set the higher numbered BAR to **Disabled**. A non-prefetchable 64-bit BAR is not supported because in a typical system, the Root Port Type 1 Configuration Space sets the maximum non-prefetchable memory window to 32 bits. The BARs can also be configured as separate 32-bit memories. Defining memory as prefetchable allows contiguous data to be fetched ahead. Prefetching memory is advantageous when 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: • Reads do not have side effects such as changing the value of the data read • Write merging is allowed
Size	N/A	Platform Designer automatically calculates the required size after you connect your components.

3.4. Device Identification Registers

Table 20. 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 21. 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. The **Maximum payload size** is 256 Bytes for the **Avalon-MM** interface and for the **Avalon-MM with DMA** interface.
Completion timeout range	**ABCD** **BCD** **ABC** **AB** **B** **A** **None**	ABCD	Indicates device function support for the optional completion timeout programmability mechanism. This mechanism allows system software to modify the completion timeout value. This field is applicable only to Root Ports and Endpoints that issue requests on their own behalf. This parameter must be set to **NONE** for the **Avalon-MM with DMA** interface. 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 22. Error Reporting

Parameter	Value	Default Value	Description
Advanced error reporting (AER)	**On/Off**	Off	When **On**, enables the Advanced Error Reporting (AER) capability.
ECRC checking	**On/Off**	Off	When **On**, enables ECRC checking. Sets the read-only value of the ECRC check capable bit in the `Advanced Error Capabilities and Control Register`. This parameter requires you to enable the AER capability.
ECRC generation	**On/Off**	Off	When **On**, enables ECRC generation capability. Sets the read-only value of the ECRC generation capable bit in the `Advanced Error Capabilities and Control Register`. This parameter requires you to enable the AER capability.
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.
Track RX completion buffer overflow on the Avalon- ST interface	**On/Off**	Off	When **On**, the core includes the `rxfx_cplbuf_ovf` output status signal to track the RX posted completion buffer overflow status

Related Information

PCI Express Base Specification Revision 3.0

3.5.3. Link Capabilities

Table 23. 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 24. 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. [6]
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 25. Slot Capabilities

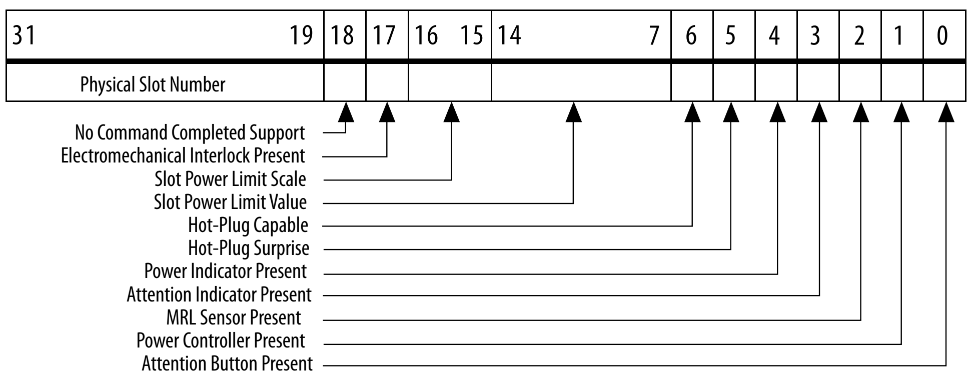
Parameter	Value	Description
Use Slot register	**On/Off**	This parameter is only supported in Root Port mode. The slot capability is required for Root Ports if a slot is implemented on the port. Slot status is recorded in the `PCI Express Capabilities` register. Defines the characteristics of the slot. You turn on this option by selecting **Enable slot capability**. Refer to the figure below for bit definitions.

continued...

(6) 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
		Not applicable for Avalon-MM DMA.
Slot power scale	0–3	Specifies the scale used for the **Slot power limit**. The following coefficients are defined: • 0 = 1.0x • 1 = 0.1x • 2 = 0.01x • 3 = 0.001x The default value prior to hardware and firmware initialization is b'00. Writes to this register also cause the port to send the `Set_Slot_Power_Limit` Message. Refer to Section 6.9 of the *PCI Express Base Specification Revision* for more information.
Slot power limit	0–255	In combination with the **Slot power scale value**, specifies the upper limit in watts on power supplied by the slot. Refer to Section 7.8.9 of the *PCI Express Base Specification* for more information.
Slot number	0-8191	Specifies the slot number.

Figure 4. Slot Capability



3.5.6. Power Management

Table 26. 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.

continued...

Parameter	Value	Description
		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. 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.

3.5.7. Vendor Specific Extended Capability (VSEC)

Table 27. 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.

3.6. Configuration, Debug, and Extension Options

Table 28. 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.

continued...

Parameter	Value	Description
		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.

3.7. PHY Characteristics

Table 29. 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.

3.8. Example Designs

Table 30. 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. 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 5. Intel Cyclone 10 GX Devices with 12 Transceiver Channels and One PCIe Hard IP Block

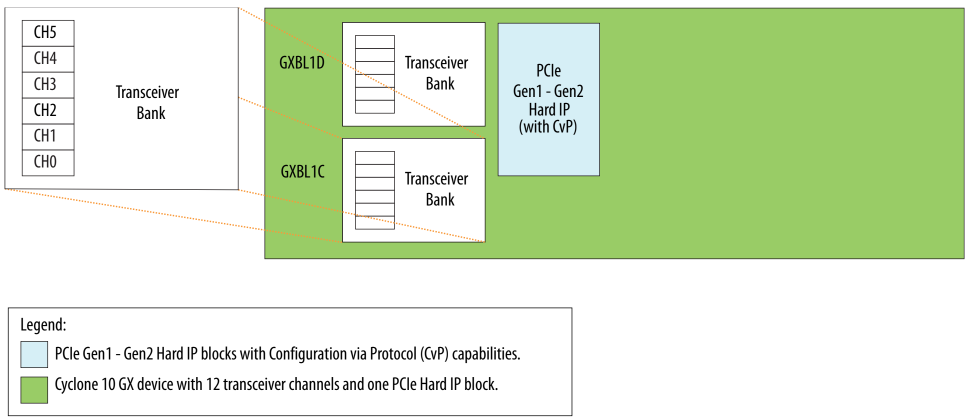


Figure 6. Intel Cyclone 10 GX Devices with 10 Transceiver Channels and One PCIe Hard IP Block

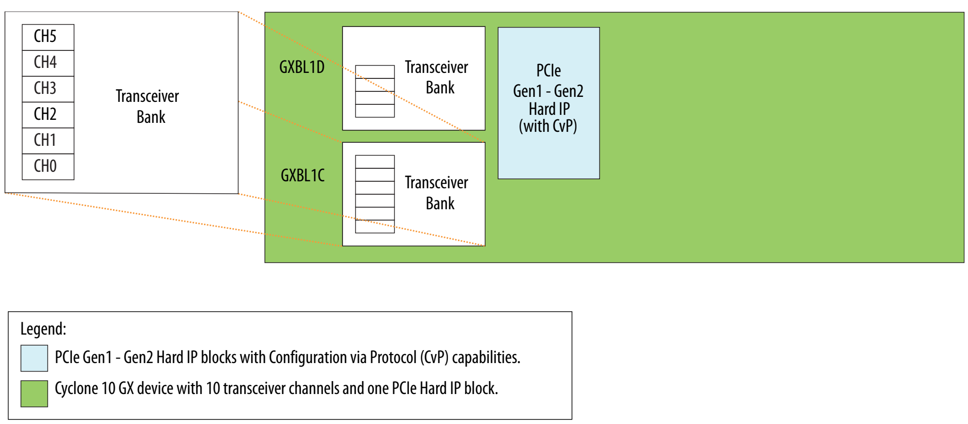
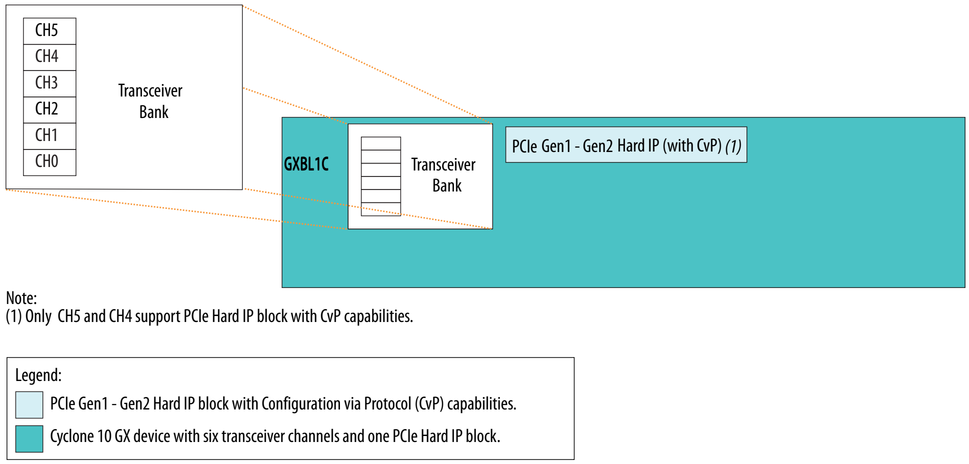


Figure 7. Intel Cyclone 10 GX Devices with 6 Transceiver Channels and One PCIe Hard IP Block



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

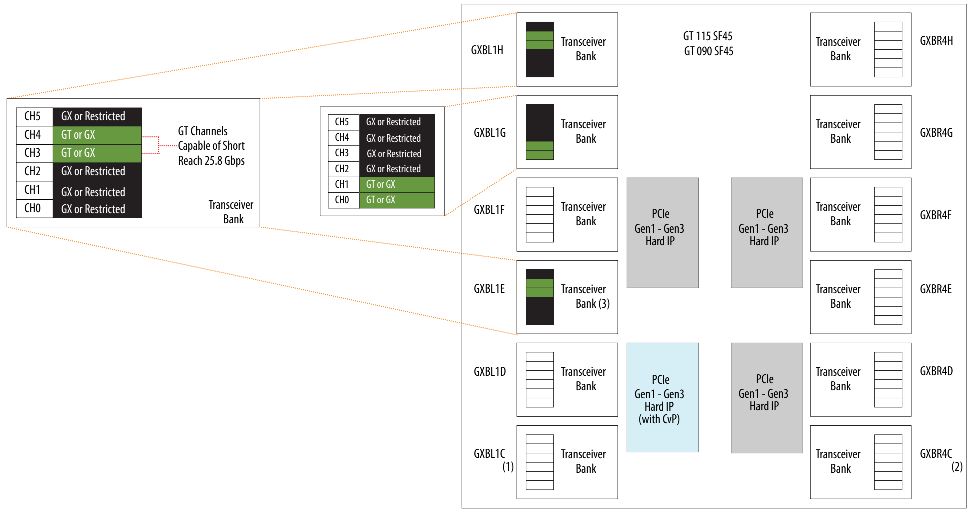
- Intel FPGA Arria 10 Transceiver PHY IP Core User Guide
  For information about the transceiver physical (PHY) layer architecture, PLLs, clock networks, and transceiver PHY IP.
- Intel Cyclone 10 GX Transceiver PHY User Guide
  For information about the transceiver PHY layer architecture, PLLs, clock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

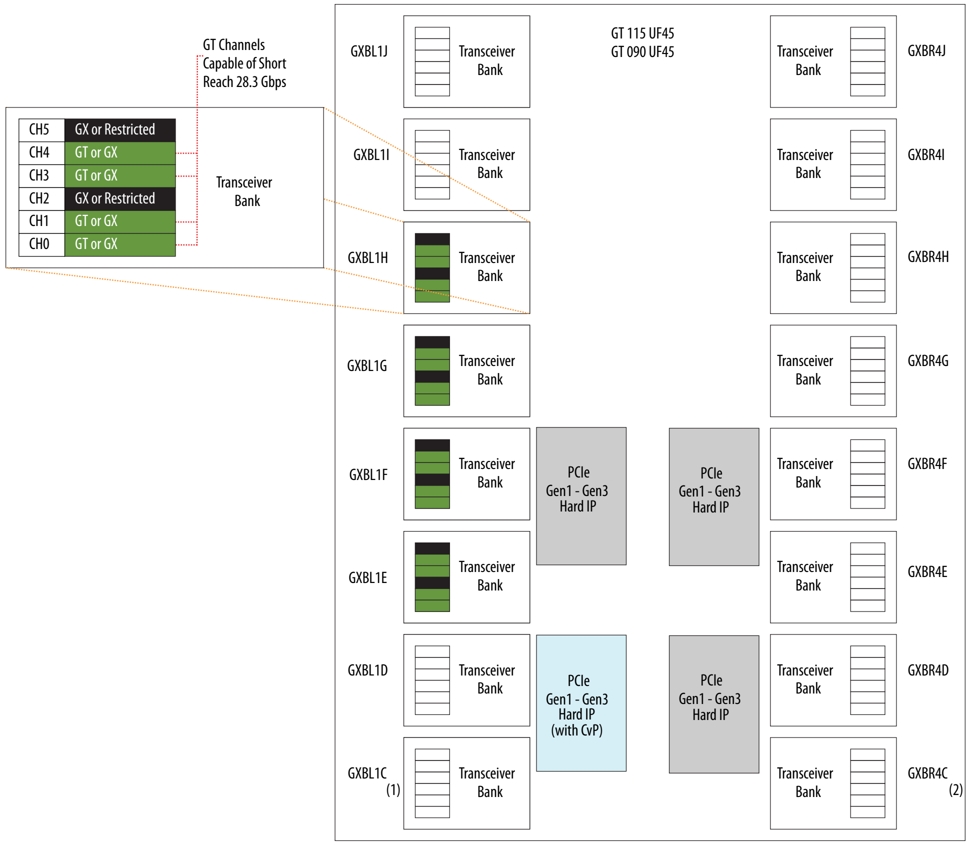
Figure 8. Intel Arria 10 Devices with 72 Transceiver Channels and Four PCIe Hard IP Blocks



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.

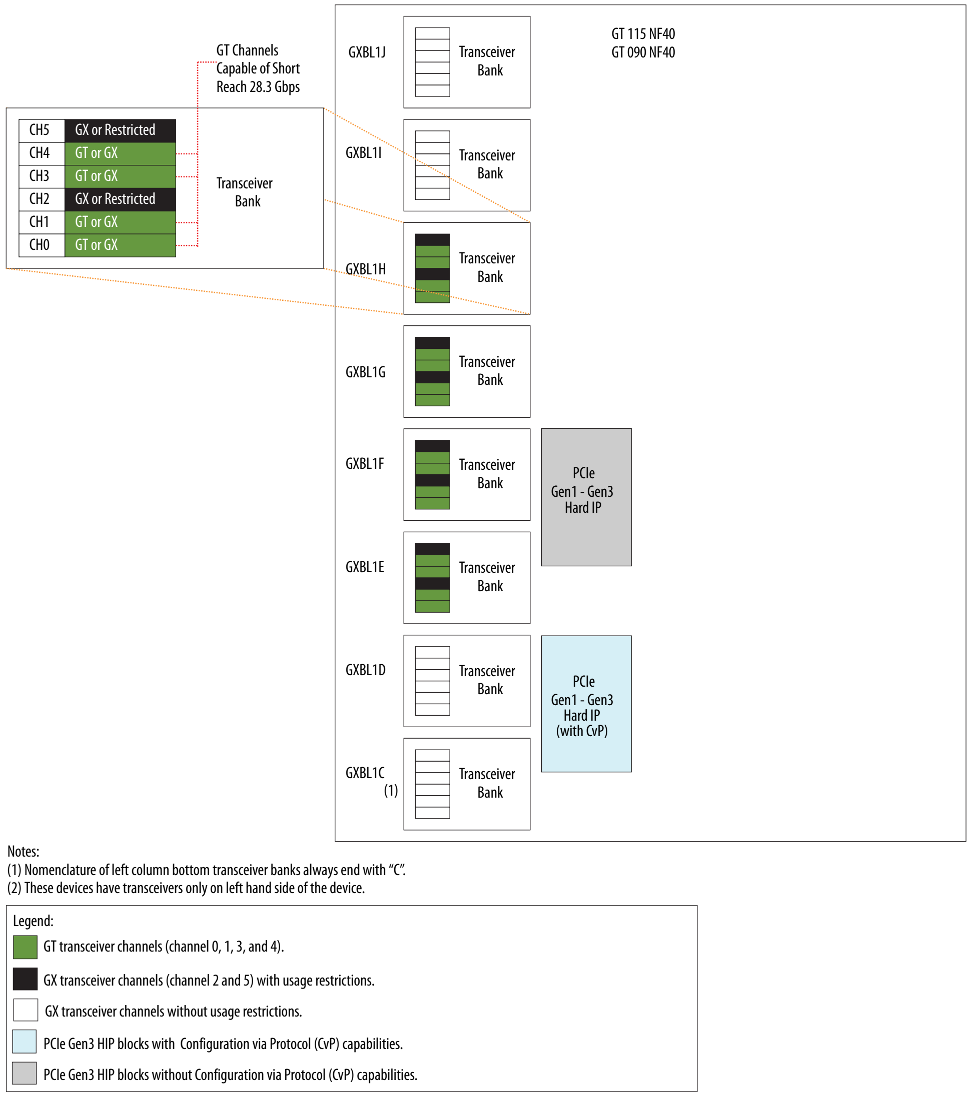
Figure 9. Intel Arria 10 Devices with 96 Transceiver Channels and Four PCIe Hard IP Blocks



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



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 11. Gen1, Gen2, and Gen3 x1 Channel and Pin Placement

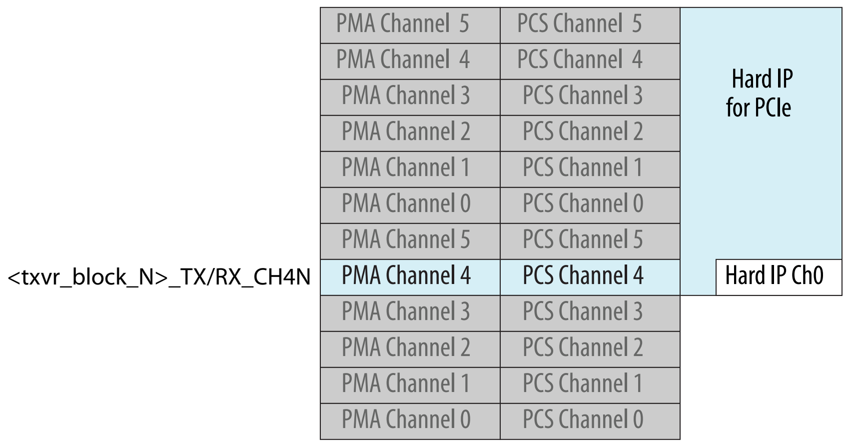


Figure 12. Gen1 Gen2, and Gen3 x2 Channel and Pin Placement

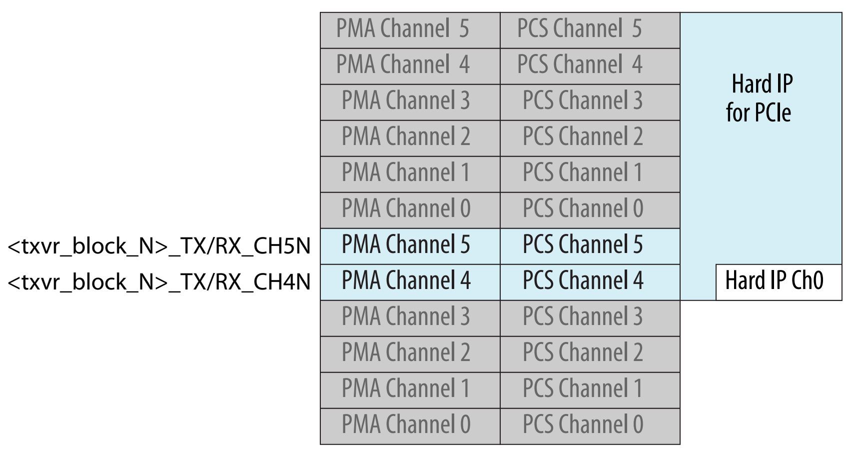


Figure 13. Gen1, Gen2, and Gen3 x4 Channel and Pin Placement

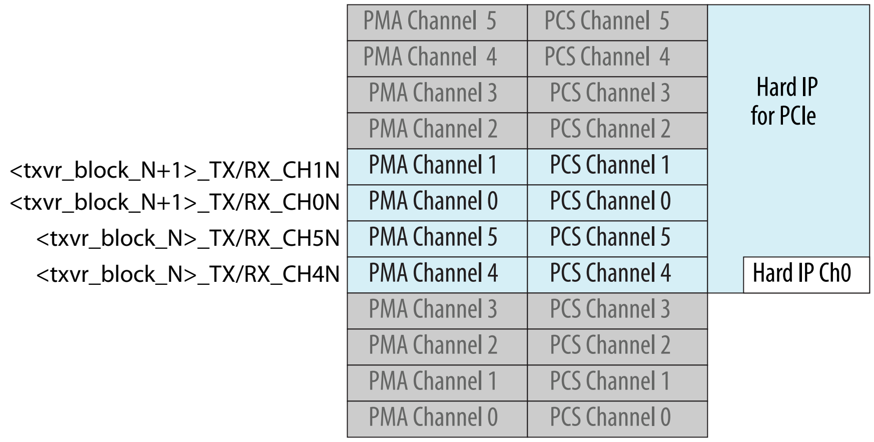
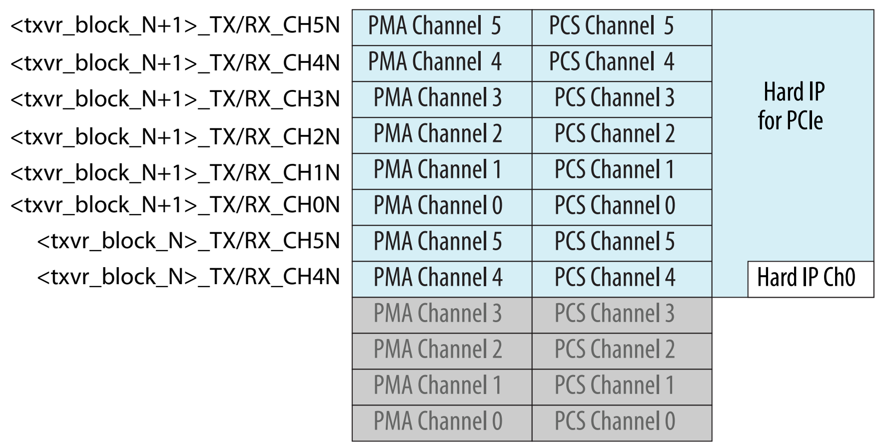


Figure 14. Gen1, Gen2, and Gen3 x8 Channel and Pin Placement



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 15. Intel Arria 10 Gen3 x1 Channel Placement

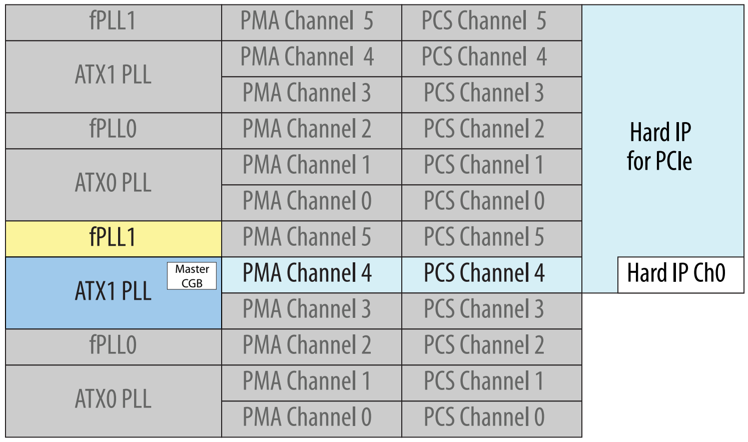


Figure 16. Intel Arria 10 Gen3 x2 Channel Placement

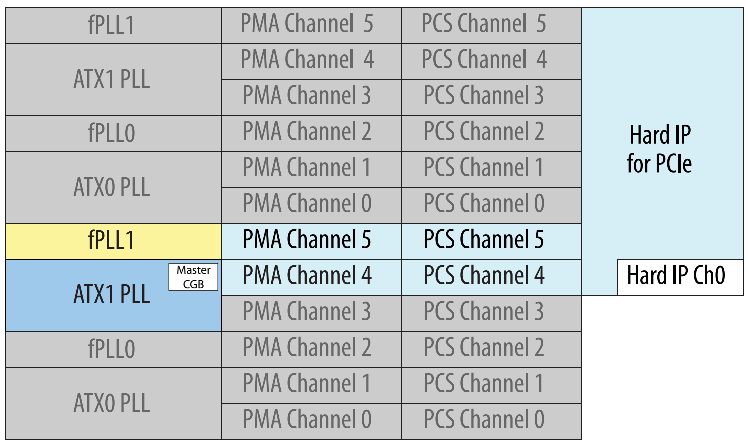


Figure 17. Intel Arria 10 Gen3 x4 Channel Placement

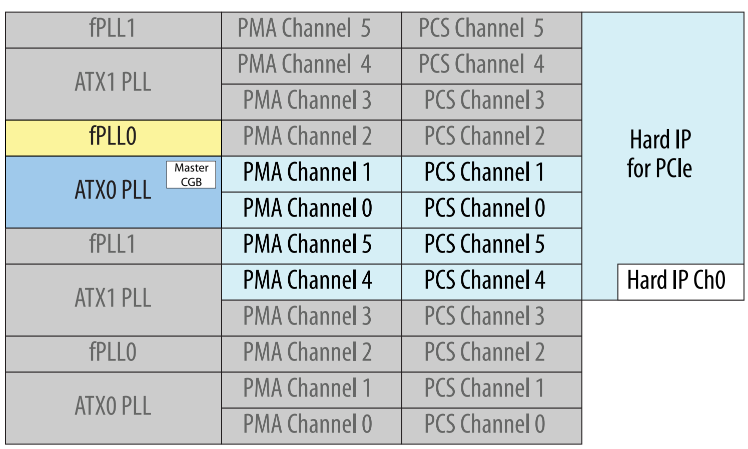
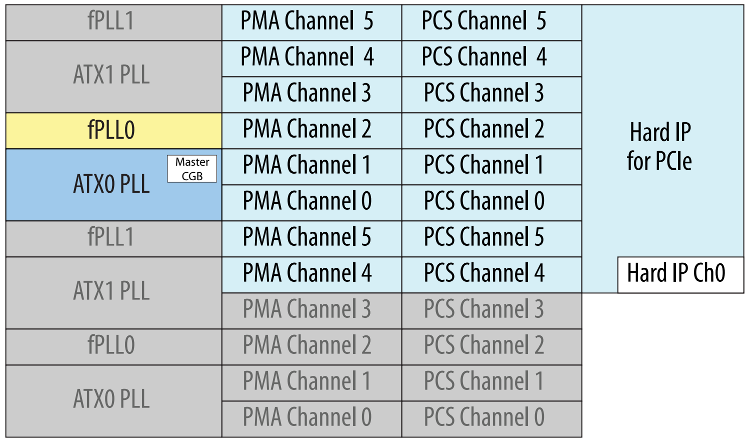


Figure 18. Gen3 x8 Channel Placement



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. IP Core Interfaces

This chapter describes the top-level signals of the Intel Arria 10 or Intel Cyclone 10 GX Hard IP for PCI Express using the Avalon-MM interface with DMA. The Avalon-MM DMA bridge includes high-performance, burst-capable Read DMA and Write DMA modules. The DMA Descriptor Controller that controls the Read DMA and Write DMA modules can be included in the Avalon-MM DMA bridge or separately instantiated. It uses 64-bit addressing, making address translation unnecessary. A separately instantiated Descriptor Controller manages the Read DMA and Write DMA modules. This variant is available for the following configurations:

- Gen1 x8
- Gen2 x4
- Gen2 x8
- Gen3 x2
- Gen3 x4
- Gen3 x8

5.1. Intel Arria 10 or Intel Cyclone 10 GX DMA Avalon-MM DMA Interface to the Application Layer

This section describes the top-level interfaces in the PCIe variant when it includes the high-performance, burst-capable read data mover and write data mover modules.

Depending on the device, the interface to the Application Layer can be 128 or 256 bits.

Figure 19. Avalon-MM DMA Bridge with Internal Descriptor Controller

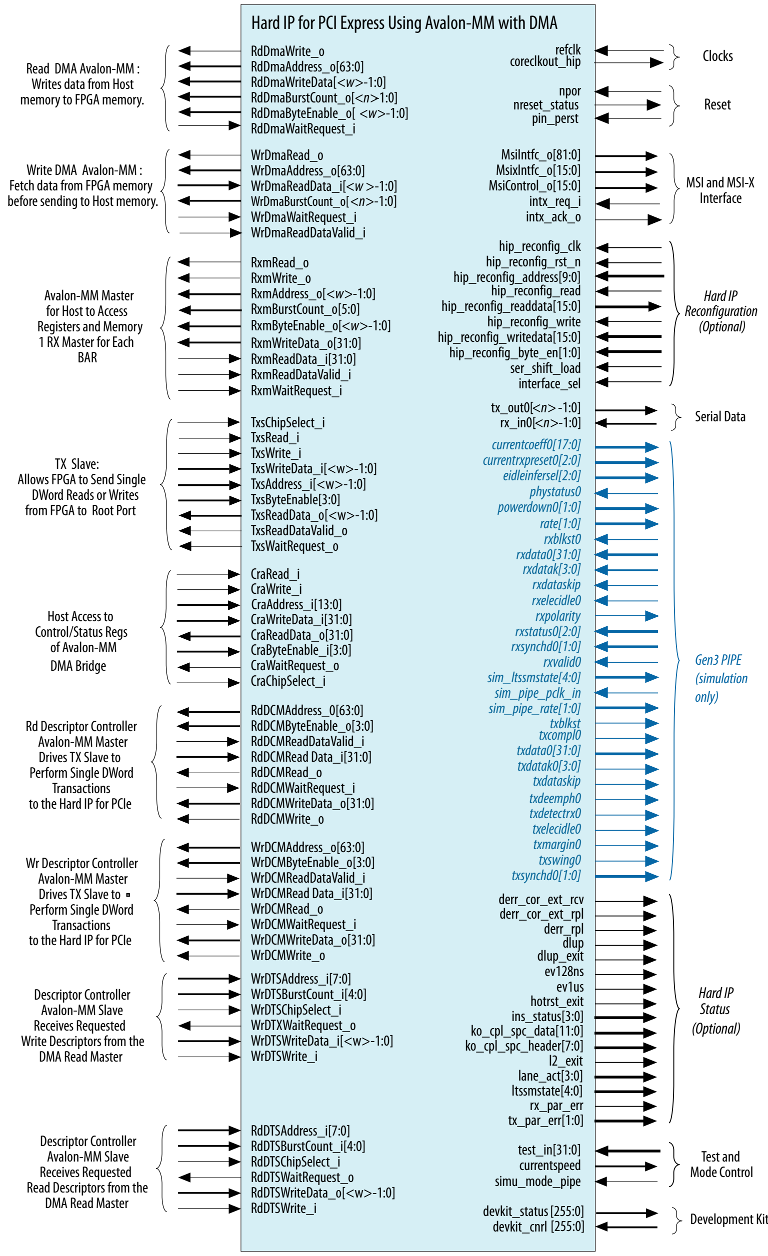
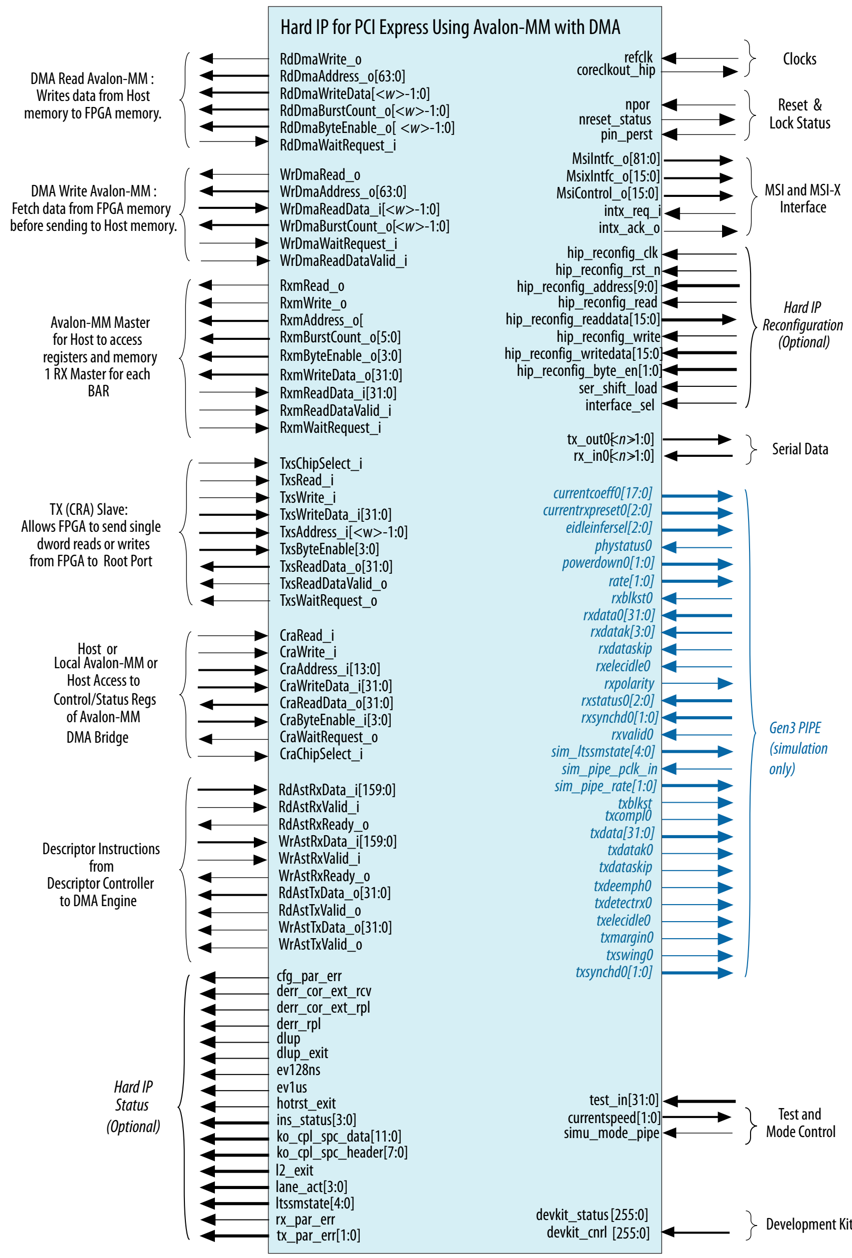


Figure 20. Avalon-MM DMA Bridge with External Descriptor Controller



This section describes the interfaces that are required to implement the DMA. All other interfaces are described in the next section, *Avalon-MM Interface to the Application Layer*.

5.1.1. Avalon-MM DMA Interfaces when Descriptor Controller Is Internally Instantiated

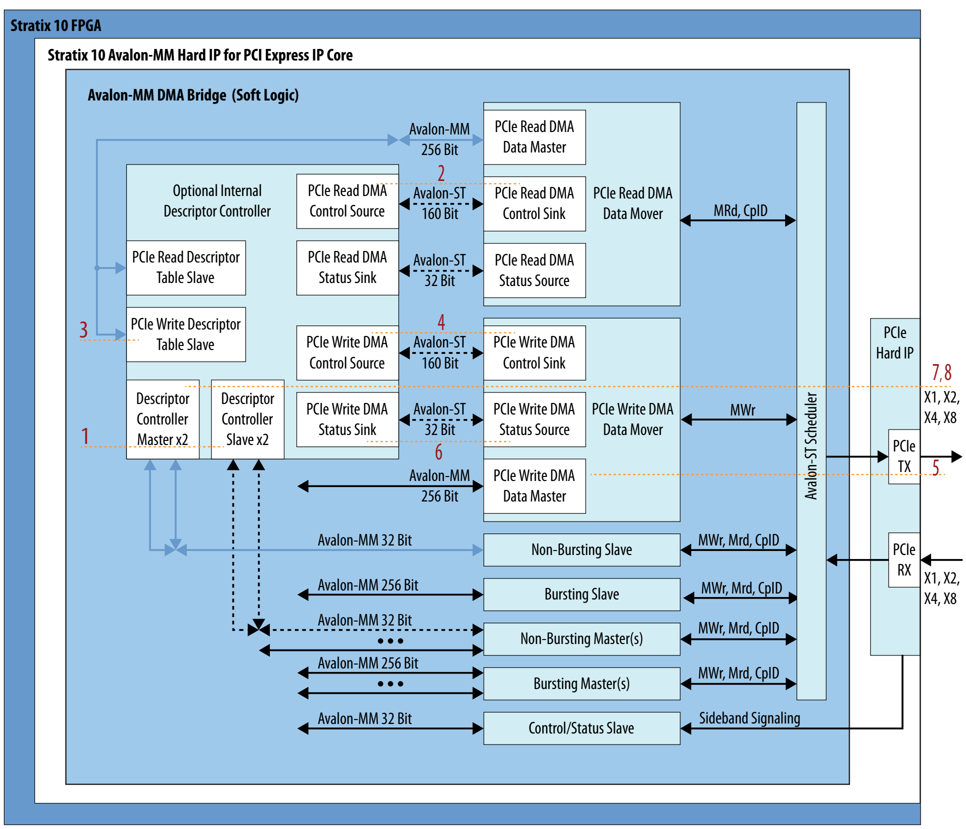
This configuration results from selecting both **Enable Avalon-MM DMA** and **Instantiate internal descriptor controller** in the component GUI.

The following figure shows the Avalon-MM DMA Bridge, implemented in soft logic. It interfaces to the Hard IP for PCIe through Avalon-ST interfaces.

In the following figure, Avalon-ST connections and the connection from the BAR0 non-bursting master to the Descriptor Controller slaves are internal. Dashed black lines show these connections. Connections between the Descriptor Controller Masters and the non-bursting slave and the connections between the Read DMA Data Master and the Descriptor Table Slaves are made in the Platform Designer. Blue lines show these connections.

Note: In the following diagrams and text descriptions, the terms Read and Write are from the system memory perspective. Thus, a Read transaction reads data from the system memory and writes it to the local memory in Avalon-MM address space. A Write transaction writes the data that was read from the local memory in Avalon-MM address space to the system memory.

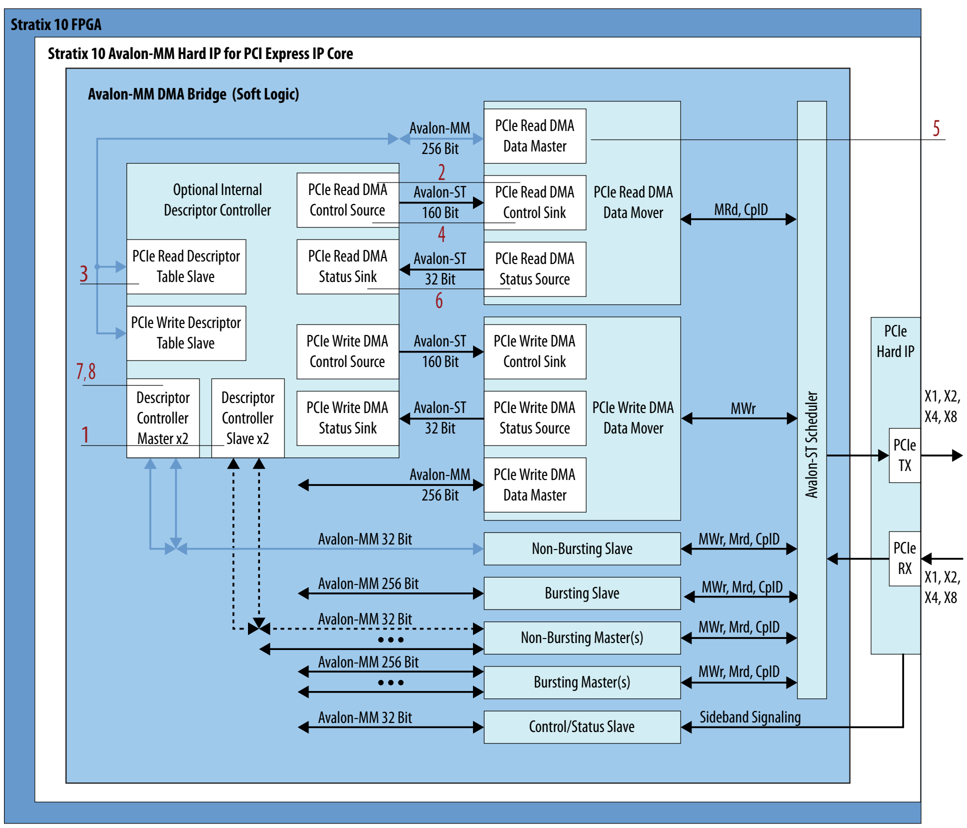
Figure 21. Avalon-MM DMA Bridge Block Diagram with Optional Internal Descriptor Controller (Showing DMA Write Flow)



The numbers in this figure describe the following steps in the DMA write flow:

1. The CPU writes registers in the Descriptor Controller Slave to start the DMA.
2. The Descriptor Controller instructs the Read Data Mover to fetch the descriptor table.
3. The Read Data Mover forwards the descriptor table to the PCIe Write Descriptor Table Slave.
4. The Descriptor Controller instructs the Write Data Mover to transfer data.
5. The Write Data Mover transfers data from FPGA to system memory.
6. The Write Data Mover notifies the Descriptor Controller of the completion of the data transfer using the done bit.
7. The Descriptor Controller Master updates the status of the descriptor table in system memory.
8. The Descriptor Controller Master sends an MSI interrupt to the host.

Figure 22. Avalon-MM DMA Bridge Block Diagram with Optional Internal Descriptor Controller (Showing DMA Read Flow)



The numbers in this figure describe the following steps in the DMA read flow:

1. The CPU writes registers in the Descriptor Controller Slave to start the DMA.
2. The Descriptor Controller instructs the Read Data Mover to fetch the descriptor table.
3. The Read Data Mover forwards the descriptor table to the PCIe Read Descriptor Table Slave.
4. The Descriptor Controller instructs the Read Data Mover to transfer data.
5. The Read Data Mover transfers data from system memory to FPGA.
6. The Read Data Mover notifies the Descriptor Controller of the completion of the data transfer using the `done` bit.
7. The Descriptor Controller Master updates the status of the descriptor table in system memory.
8. The Descriptor Controller Master sends an MSI interrupt to the host.

When the optional Descriptor Controller is included in the bridge, the Avalon-MM bridge includes the following Avalon interfaces to implement the DMA functionality:

- PCIe Read DMA Data Master (rd_dma): This is a 256-bit wide write only Avalon-MM master interface which supports bursts of up to 16 cycles with the `rd_dma*` prefix. The Read Data Mover uses this interface to write at high throughput the blocks of data that it has read from the PCIe system memory space. This interface writes descriptors to the Read and Write Descriptor table slaves and to any other Avalon-MM connected slaves interfaces.
- PCIe Write DMA Data Master (wr_dma): This read-only interface transfers blocks of data from the Avalon-MM domain to the PCIe system memory space at high throughput. It drives read transactions on its bursting Avalon-MM master interface. It also creates PCIe Memory Write (MWr) TLPs with data payload from Avalon-MM reads. It forwards the MWr TLPs to the Hard IP for transmission on the link. The Write Data Mover module decomposes the transfers into the required number of Avalon-MM burst read transactions and PCIe MWr TLPs. This is a bursting, 256-bit Avalon-MM interface with the `wr_dma` prefix.
- PCIe Read Descriptor Table Slave (rd_dts): This is a 256-bit Avalon-MM slave interface that supports write bursts of up to 16 cycles. The PCIe Read DMA Data Master writes descriptors to this table. This connection is made outside the DMA bridge because the Read Data Mover also typically connects to other Avalon-MM slaves. The prefix for this interface is `rd_dts`.

- PCIe Write Descriptor Table Slave (wr_dts): This is a 256-bit Avalon-MM slave interface that supports write bursts of up to 16 cycles. The PCIe Read DMA Data Master writes descriptors to this table. The PCIe Read DMA Data Master must connect to this interface outside the DMA bridge because the bursting master interface may also need to be connected to the destination of the PCIe Read Data Mover. The prefix for this interface is `wr_dts`.
- Descriptor Controller Master (DCM): This is a 32-bit, non-bursting Avalon-MM master interface with write-only capability. It controls the non-bursting Avalon-MM slave that transmits single DWORD DMA status information to the host. The prefixes for this interface are `wr_dcm and rd_dcm.`
- Descriptor Controller Slave (DCS): This is a 32-bit, non-bursting Avalon-MM slave interface with read and write access. The host accesses this interface through the BAR0 Non-Bursting Avalon-MM Master, to program the Descriptor Controller.

  Note: This is not a top-level interface of the Avalon-MM Bridge. Because it connects to BAR0, you cannot use BAR0 to access any other Avalon-MM slave interface.

5.1.2. Read Data Mover

The Read Data module sends memory read TLPs. It writes the completion data to an external Avalon-MM interface through the high throughput Read Master port. This data mover operates on descriptors the IP core receives from the DMA Descriptor Controller.

The Read DMA Avalon-MM Master interface performs the following functions:

1. Provides the Descriptor Table to the Descriptor Controller

The Read Data Mover sends PCIe system memory read requests to fetch the descriptor table from PCIe system memory. This module then writes the returned descriptor entries in to the Descriptor Controller FIFO using this Avalon-MM interface.

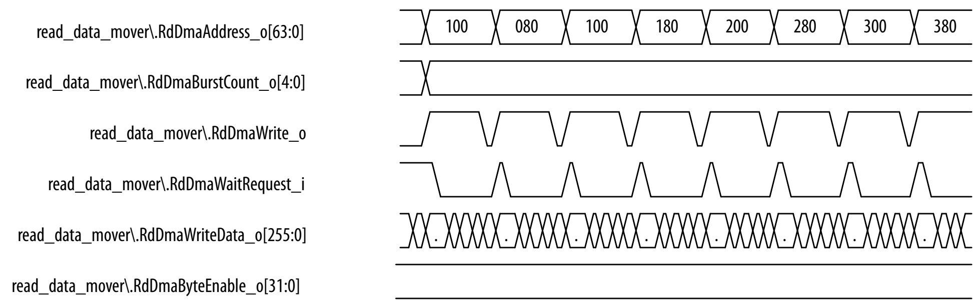
2. Writes Data to Memory Located in Avalon-MM Space

After a DMA Read finishes fetching data from the source address in PCIe system memory, the Read Data Mover module writes the data to the destination address in Avalon-MM address space via this interface.

Table 31. Read DMA 256-Bit Avalon-MM Master Interface

Signal Name	Direction	Description
`RdDmaWrite_o`	Output	When asserted, indicates that the Read DMA module is ready to write read completion data to a memory component in the Avalon-MM address space.
`RdDmaAddress_o[63:0]`	Output	Specifies the write address in the Avalon-MM address space for the read completion data.
`RdDmaWriteData_o[127 or 255:0]`	Output	The read completion data to be written to the Avalon-MM address space.
`RdDmaBurstCount_o[4:0] or [5:0]`	Output	Specifies the burst count in 128- or 256-bit words. This bus is 5 bits for the 256-bit interface. It is 6 bits for the 128-bit interface.
`RdDmaByteEnable_o[15 or 31:0]`	Output	Specifies which DWORDs are valid.
`RdDmaWaitRequest_i`	Input	When asserted, indicates that the memory is not ready to receive data.

Figure 23. Read DMA Avalon-MM Master Writes Data to FPGA Memory



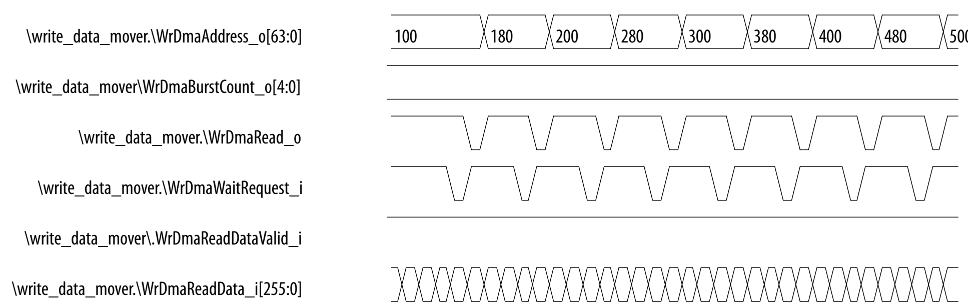
5.1.3. Write DMA Avalon-MM Master Port

The Write Data Mover module fetches data from the Avalon-MM address space using this interface before issuing memory write requests to transfer data to PCIe system memory.

Table 32. DMA Write 256-Bit Avalon-MM Master Interface

Signal Name	Direction	Description
`WrDMARead_o`	Output	When asserted, indicates that the Write DMA module is reading data from a memory component in the Avalon-MM address space to write to the PCIe address space.
`WrDmaAddress_o[63:0]`	Output	Specifies the address for the data to be read from a memory component in the Avalon-MM address space .
`WrDmaReadData_i[127 or 255:0]`	Input	Specifies the completion data that the Write DMA module writes to the PCIe address space.
`WrDmaBurstCount_o[4:0]or[5:0]`	Output	Specifies the burst count in 128- or 256-bit words. This bus is 5 bits for the 256-bit interface. It is 6 bits for the 128-bit interface
`WrDmaWaitRequest_i`	Input	When asserted, indicates that the memory is not ready to be read.
`WrDmaReadDataValid_i`	Input	When asserted, indicates that `WrDmaReadData_i` is valid.

Figure 24. Write DMA Avalon-MM Master Reads Data from FPGA Memory



5.1.4. RX Master Module

The RX Master module translates read and write TLPs received from the PCIe link to Avalon-MM requests for Platform Designer components connected to the interconnect. This module allows other PCIe components, including host software, to access other Avalon-MM slaves connected in the Platform Designer system.

If burst mode is not enabled, the RX Master module only supports 32-bit read or write request. All other requests received from the PCIe link are considered a violation of this device's programming model, and are therefore handled with the PCIe Completer Abort status. You can enable burst mode for BAR2 using 32-bit addressing or BAR2 and BAR3 using 64-bit addressing. When enabled, the module supports dword, burst read, or write requests. When the Descriptor Controller is internally instantiated, the RX Master for BAR0 is used internally and not available for other uses.

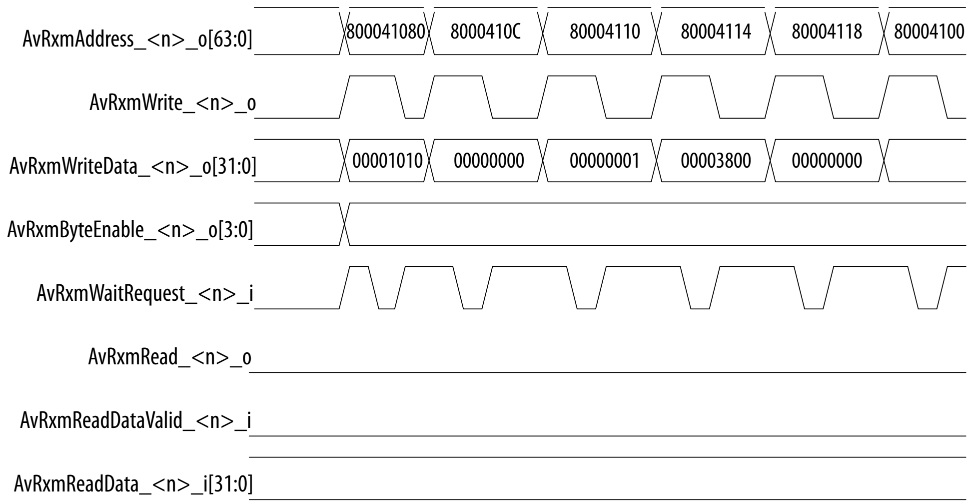
Note: When you enable burst mode for BAR2, Flush reads (i.e. reads with all byte enables set to 0) are not supported.

Table 33. RX Master Control Interface Ports for BAR Access

Each BAR has one corresponding RX Master Control interface. In this table, *<n>* is the BAR number.

Signal Name	Direction	Description
`RxmRead_<n>_o`	Output	When asserted, indicates an Avalon-MM read request.
`RxmWrite_<n>_o`	Output	When asserted, indicates an Avalon-MM write request.
`RxmAddress_<n>_o[<w>-1:0]`	Output	Specifies the Avalon-MM byte address. Because all addresses are byte addresses, the meaningful bits of this address are [*<w>*-1:2]. Bits 1 and 0 have a value of 0. *<w>* can be 32 or 64.
`RxmBurstCount_<n>_o[5:0]`	Output	Specifies the burst count in dwords (32 bits). This optional signal is available for BAR2 when you turn on **Enable burst capabilities for RXM BAR2 ports**.
`RxmByteEnable_<n>_o[<w>-1:0]`	Output	Specifies the valid bytes of data to be written. *<w>* has the following values: • 4: for the non-bursting RX Master • 32: for the bursting 128-bit Avalon-MM interface • 64: for the bursting 256-bit Avalon-MM interface
`RxmDataWrite_<n>_o[<w>-1:0]`	Output	Specifies the Avalon-MM write data. *<w>* has the following values: • 32: for the non-bursting RX Master • 128: for the bursting 128-bit Avalon-MM interface • 256: for the bursting 256-bit Avalon-MM interface
`RxmReadData_<n>_i[<w>-1:0]`	Input	Specifies the Avalon-MM read data. *<w>* has the following values: • 32: for the non-bursting RX Master • 128: for the bursting 128-bit Avalon-MM interface • 256: for the bursting 256-bit Avalon-MM interface
`RxmReadDataValid_<n>_i[31:0]`	Input	`When asserted, indicates that RxmReadData_i[31:0]is valid.`
`RxmWaitRequest_<n>_i`	Input	When asserted indicates that the control register access Avalon-MM slave port is not ready to respond.

Figure 25. RXM Master Writes to Memory in the Avalon-MM Address Space



5.1.5. Non-Bursing Slave Module

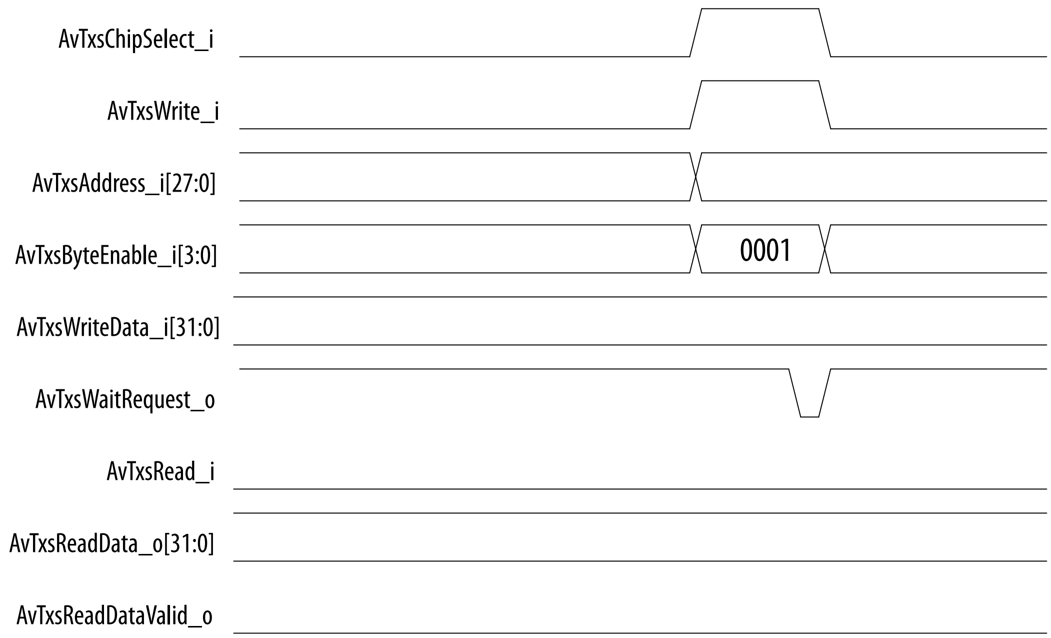
The TX Slave module translates Avalon-MM read and write requests to PCI Express TLPs.

The slave module supports a single outstanding non-bursting request. It typically sends status updates to the host. This is a 32-bit Avalon-MM slave interface.

Table 34. TX Slave Control

Signal Name	Direction	Description
`TxsChipSelect_i`	Input	When asserted, indicates that this slave interface is selected. When `txs_chipselect_i` is deasserted, `txs_read_i` and `txs_write_i` signals are ignored.
`TxsRead_i`	Input	When asserted, specifies a Avalon-MM Ignored when the chip select is deasserted.
`TxsWrite_i`	Input	When asserted, specifies a Avalon-MM Ignored when the chip select is deasserted.
`TxsWriteData_i[31:0]`	Input	Specifies the Avalon-MM data for a write command.
`TxsAddress_i[<w>-1:0]`	Input	Specifies the Avalon-MM byte address for the read or write command. The width of this address bus is specified by the parameter **Address width of accessible PCIe memory space**.
`TxsByteEnable_i[3:0]`	Input	Specifies the valid bytes for a write command.
`TxsReadData_o[31:0]`	Output	Drives the read completion data.
`TxsReadDataValid_o`	Output	When asserted, indicates that read data is valid.
`TxsWaitRequest_o`	Output	When asserted, indicates that the Avalon-MM slave port is not ready to respond to a read or write request. The non-bursting Avalon-MM slave may assert`TxsWaitRequest_o` during idle cycles. An Avalon-MM master may initiate a transaction when `TxsWaitRequest_o` is asserted and wait for that signal to be deasserted.

Figure 26. Non-Bursting Slave Interface Sends Status to Host



5.1.6. 32-Bit Control Register Access (CRA) Slave Signals

The CRA interface provides access to the control and status registers of the Avalon-MM bridge. This interface has the following properties:

- 32-bit data bus
- Supports a single transaction at a time
- Supports single-cycle transactions (no bursting)

Note: When Intel Stratix 10 is in Root Port mode, and the application logic issues a CfgWr or CfgRd via the CRA interface, it needs to fill the Tag field in the TLP Header with the value 0x10 to ensure that the corresponding Completion gets routed to the CRA interface correctly. If the application logic sets the Tag field to some other value, the Intel Stratix 10 Avalon-MM Hard IP for PCIe IP Core does not overwrite that value with the correct value.

Table 35. Avalon-MM CRA Slave Interface Signals

Signal Name	Direction	Description
CraRead_i	Input	Read enable.
CraWrite_i	Input	Write request.
CraAddress_i[13:0]	Input	An address space of 16 KB 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 this signal to 4'b0011.

continued...

Signal Name	Direction	Description
		An address space of 32 KB 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 this signal to 4'b0011.
`CraWriteData_i[31:0]`	Input	Write data. The current version of the CRA slave interface is read-only. Including this signal as part of the Avalon-MM interface, makes future enhancements possible.
`CraReadData_o[31:0]`	Output	Read data lines.
`CraByteEnable_i[3:0]`	Input	Byte enable.
`CraWaitRequest_o`	Output	Wait request to hold off additional requests.
`CraChipSelect_i`	Input	Chip select signal to this slave.
`CraIrq_o`	Output	Interrupt request. A port request for an Avalon-MM interrupt.

Related Information

DMA Descriptor Controller Registers on page 80

5.1.7. Avalon-ST Descriptor Control Interface when Instantiated Separately

After fetching multiple descriptor entries from the Descriptor Table in the PCIe system memory, the Descriptor Controller uses its Avalon-ST Descriptor source interface to transfer 160-bit Descriptors to the Read or Write DMA Data Movers.

Table 36. Avalon-ST Descriptor Sink Interface

This interface sends instructions from Descriptor Controller to Read DMA Engine.

Signal Name	Direction	Description
`RdAstRxData_i[159:0]`	Input	Specifies the descriptors for the Read DMA module. Refer to *DMA Descriptor Format* table below for bit definitions.
`RdAstRxValid_i`	Input	When asserted, indicates that the data is valid.
`RdAstRxReady_o`	Output	When asserted, indicates that the Read DMA read module is ready to receive a new descriptor. The ready latency is 3 cycles. Consequently, interface can accept data 3 cycles after ready is asserted.

Table 37. Avalon-ST Descriptor Sink Interface

This interface sends instructions from Descriptor Controller to Write DMA Engine.

Signal Name	Direction	Description
`WrAstRxData_i[159:0]`	Input	Specifies the descriptors for the Write DMA module. Refer to *DMA Descriptor Format* table below for bit definitions.
`WrAstRxValid_i`	Input	When asserted, indicates that the data is valid.
`WrAstRxReady_o`	Output	When asserted, indicates that the Write DMA module engine is ready to receive a new descriptor. The ready latency for this signal is 3 cycles. Consequently, interface can accept data 3 cycles after ready is asserted.

5.1.7.1. Avalon-ST Descriptor Status Interface

When DMA module completes the processing for one Descriptor Instruction, it returns DMA Status to the Descriptor Controller via the following interfaces.

Table 38. Read DMA Status Interface from Read DMA Engine to Descriptor Controller

Signal Name	Direction	Description
`RdAstTxData_o[31:0]`	Output	Drives status information to the Descriptor Controller component. Refer to *DMA Status Bus* table below for more information
`RdAstTxValid_o`	Output	When asserted, indicates that the data is valid.

Table 39. Write DMA Status Interface from Write DMA Engine to Descriptor Controller

Signal Name	Direction	Description
`WrAstTxData_o[31:0]`	Output	Drives status information to the Descriptor Controller component. Refer to *DMA Status Bus* table below for more information about this bus.
`WrAstTxValid_o`	Output	When asserted, indicates that the data is valid.

Table 40. DMA Descriptor Format

Bits	Name	Description
[31:0]	`Source Low Address`	Low-order 32 bits of the DMA source address. The address boundary must align to the 32 bits so that the 2 least significant bits are 2'b00. For the Read DMA module, the source address is the PCIe domain address. For the Write DMA module, the source address is the Avalon-MM domain address.
[63:32]	`Source High Address`	High-order 32 bits of the source address.
[95:64]	`Destination Low Address`	Low-order 32 bits of the DMA destination address. The address boundary must align to the 32 bits so that the 2 least significant bits have the value of 2'b00. For the Read DMA module, the destination address is the Avalon-MM domain address. For the Write DMA module the destination address is the PCIe domain address.
[127:96]	`Destination High Address`	High-order 32 bits of the destination address.
[145:128]	`DMA Length`	Specifies DMA length in dwords. The length must be greater than 0. The maximum length is 1 MB - 4 bytes.
[153:146]	`DMA Descriptor ID`	Specifies up to 128 descriptors.
[158:154]	`Reserved`	—
[159]	`Immediate Write`	When set to 1'b1, the Write DMA Engine performs an `Immediate Write`. The `Immediate Write` provides a fast mechanism to send a Write TLP upstream. The descriptor stores the 32-bit payload, replacing the `Source Low Address` field of the descriptor.

5.1.7.2. Avalon-ST Descriptor Status Sources

Read DMA and Write DMA modules report status to the Descriptor Controller on the `RdDmaTxData_o[31:0]` or `WrDmaTxData_o[31:0]` bus when a descriptor completes successfully.

The following table shows the mappings of the triggering events to the DMA descriptor status bus:

Table 41. DMA Status Bus

Bits	Name	Description
[31:9]	Reserved	—
[8]	Done	When asserted, a single DMA descriptor has completed successfully.
[7:0]	Descriptor ID	The ID of the descriptor whose status is being reported.

5.1.8. Descriptor Controller Interfaces when Instantiated Internally

The Descriptor Controller controls the Read DMA and Write DMA Data Movers. It provides a 32-bit Avalon-MM slave interface to control and manage data flow from PCIe system memory to Avalon-MM memory and in the reverse direction.

The Descriptor Controller includes two, 128-entry FIFOs to store the read and write descriptor tables. The Descriptor Controller forwards the descriptors to the Read DMA and Write DMA Data Movers.

The Data Movers send completion status to the Read Descriptor Controller and Write Descriptor Controller. The Descriptor Controller forwards status and MSI to the host using the TX slave port.

5.1.8.1. Read Descriptor Controller Avalon-MM Master interface

The Read Descriptor Controller Avalon-MM master interface drives the non-bursting Avalon-MM slave interface. The Read Descriptor Controller uses this interface to write descriptor status to the PCIe domain and possibly to MSI when MSI messages are enabled. This Avalon-MM master interface is only available for variants with the internally instantiated Descriptor Controller.

By default MSI interrupts are enabled. You specify the **Number of MSI messages requested** on the **MSI** tab of the parameter editor. The MSI Capability Structure is defined in *Section 6.8.1 MSI Capability Structure* of the *PCI Local Bus Specification*.

Table 42. Read Descriptor Controller Avalon-MM Master interface

Signal Name	Direction	Description
RdDCMAddress_o[63:0]	Output	Specifies the descriptor status table or MSI address.
RdDCMByteEnable_o[3:0]	Output	Specifies which data bytes are valid.
RdDCMReadDataValid_i	Input	When asserted, indicates that the read data is valid.
RdDCMReadData_i[31:0]	Input	Specifies the read data of the descriptor status table entry addressed.
RdDCMRead_o	Output	When asserted, indicates a read transaction. Currently, this is a write-only interface so that this signal never asserts.
RdDCMWaitRequest_i	Input	When asserted, indicates that the connected Avalon-MM slave interface is busy and cannot accept a transaction.
RdDCMWriteData_o[31:0]	Output	Specifies the descriptor status or MSI data..
RdDCMWrite_o	Output	When asserted, indicates a write transaction.

Related Information

- MSI and MSI-X Capabilities on page 28
- PCI Local Bus Specification

5.1.8.2. Write Descriptor Controller Avalon-MM Master Interface

The Avalon-MM Descriptor Controller Master interface is a 32-bit single-DWORD master with wait request support. The Write Descriptor Controller uses this interface to write status back to the PCI-Express domain and possibly MSI when MSI messages are enabled. This Avalon-MM master interface is only available for the internally instantiated Descriptor Controller.

By default MSI interrupts are enabled. You specify the **Number of MSI messages requested** on the **MSI** tab of the parameter editor. The MSI Capability Structure is defined in *Section 6.8.1 MSI Capability Structure* of the *PCI Local Bus Specification*.

Table 43. Write Descriptor Controller Avalon-MM Master interface

Signal Name	Direction	Description
WrDCMAddress_o[63:0]	Output	Specifies the descriptor status table or MSI address.
WrDCMByteEnable_o[3:0]	Output	Specifies which data bytes are valid.
WrDCMReadDataValid_i	Input	When asserted, indicates that the read data is valid.
WrRdDCMReadData_i[31:0]	Output	Specifies the read data for the descriptor status table entry addressed.
WrDCMRead_o	Output	When asserted, indicates a read transaction.
WrDCMWaitRequest_i	Input	When asserted, indicates that the Avalon-MM slave device is not ready to respond.
WrDCMWriteData_o[31:0]	Output	Specifies the descriptor status table or MSI address.
WrDCMWrite_o	Output	When asserted, indicates a write transaction.

Related Information

- MSI and MSI-X Capabilities on page 28
- PCI Local Bus Specification

5.1.8.3. Read Descriptor Table Avalon-MM Slave Interface

This interface is available when you select the internal Descriptor Controller. It receives the Read DMA descriptors which are fetched by the Read Data Mover. Connect the interface to the Read DMA Avalon-MM master interface.

Table 44. Read Descriptor Table Avalon-MM Slave Interface

Signal Name	Direction	Description
RdDTSAddress_i[7:0]	Input	Specifies the descriptor table address.
RdDTSBurstCount_i[4:0] or [5:0]	Input	Specifies the burst count of the transaction in words.
RdDTSChipSelect_i	Input	When asserted, indicates that the read targets this slave interface.

continued...

Signal Name	Direction	Description
RdDTSWriteData_i[255:0] or [127:0]	Input	Specifies the descriptor.
RdDTSWrite_i	Input	When asserted, indicates a write transaction.
RdDTSWaitRequest_o	Output	When asserted, indicates that the Avalon-MM slave device is not ready to respond.

5.1.8.4. Write Descriptor Table Avalon-MM Slave Interface

This interface is available when you select the internal Descriptor Controller. This interface receives the Write DMA descriptors which are fetched by Read Data Mover. Connect the interface to the Read DMA Avalon-MM master interface.

Table 45. Write Descriptor Table Avalon-MM Slave Interface

Signal Name	Direction	Description
WrDTSAddress_i[7:0]	Input	Specifies the descriptor table address.
WrDTSBurstCount_i[4:0]	Input	Specifies the burst count of the transaction in words.
WrDTSChipSelect_i	Input	When asserted, indicates that the write is for this slave interface.
WrDTSWaitRequest_o	Output	When asserted, indicates that this interface is busy and is not ready to respond.
WrDTSWriteData_i[255:0] or [127:0]	Input	Drives the descriptor table entry data.
WrDTSWrite_i	Input	When asserted, indicates a write transaction.

5.2. Clock Signals

Table 46. 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 98

5.3. Reset, Status, and Link Training Signals

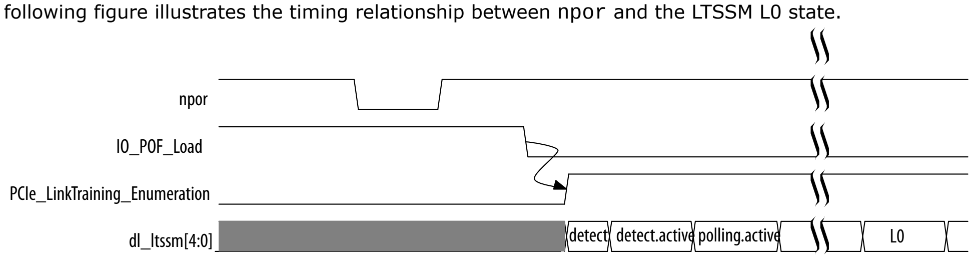
Refer to *Reset and Clocks* for more information about the reset sequence and a block diagram of the reset logic.

Table 47. 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 and Clocks*.
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 *Intel Arria 10 CvP Initialization and over PCI Express User Guide*. Intel Arria 10 devices can have up to 4 instances of the Hard IP for PCI Express IP Core. Intel Cyclone 10 GX devices can have a single instances of the Hard IP for PCI Express IP Core. Each instance has its own pin_perst signal. *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 27. 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.

Table 48. Status and Link Training Signals

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. (7)
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. (7)
derr_rpl	Output	Indicates an uncorrectable error in the retry buffer. This signal is for debug only. (7)
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 1µs 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[11:0]	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[7:0]	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.
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.

continued...

(7) Intel does not rigorously test or verify debug signals. Only use debug signals to observe behavior. Do not use debug signals to drive custom logic.

Signal	Direction	Description
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 • 11111: Recovery.Equalization, Done
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, refer to *Uncorrectable Internal Error Status Register*. 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. Reset the IP core when this error is detected. Contact Intel technical support 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
- Intel Arria 10 CvP Initialization and Partial Reconfiguration over PCI Express User Guide
- Intel Cyclone 10 GX CvP Initialization over PCI Express User Guide

5.4. MSI Interrupts for Endpoints

The MSI interrupt notifies the host when a DMA operation has completed. After the host receives this interrupt, it can poll the DMA read or write status table to determine which entry or entries have the done bit set. This mechanism allows host software to avoid continuous polling of the status table done bits. Use this interface to receive information required to generate MSI or MSI-X interrupts to the Root Port via the TX Slave interface.

Table 49. MSI Interrupt

Signal	Direction	Description
`MSIIntfc_o[81:0]`	Output	This bus provides the following MSI address, data, and enabled signals: • `MSIIntfc_o[81]`: Master enable • `MSIIntfc_o[80]`: MSI enable • `MSIIntfc_o[79:64]`: MSI data • `MSIIntfc_o[63:0]`: MSI address
`MSIXIntfc_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: • `MSIXIntfc_o[15]`: Enable • `MSIXIntfc_o[14]`: Mask • `MSIXIntfc_o[13:11]`: Reserved • `MSIXIntfc_o[10:0]`: Table size
`MSIControl_o[15:0]`	Output	Provides system software control of the MSI messages 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: • `MSIControl_o[15:9]`: Reserved • `MSIControl_o[8]`: Per-Vector Masking Capable • `MSIControl_o[7]`: 64-Bit Address Capable • `MSIControl_o[6:4]`: Multiple Message Enable • `MSIControl_o[3:1]`: MSI Message Capable • `MSIControl_o[0]`: MSI Enable
`intx_req_i`	Input	Legacy interrupt request.
`intx_ack_o`	Output	Legacy interrupt acknowledge.

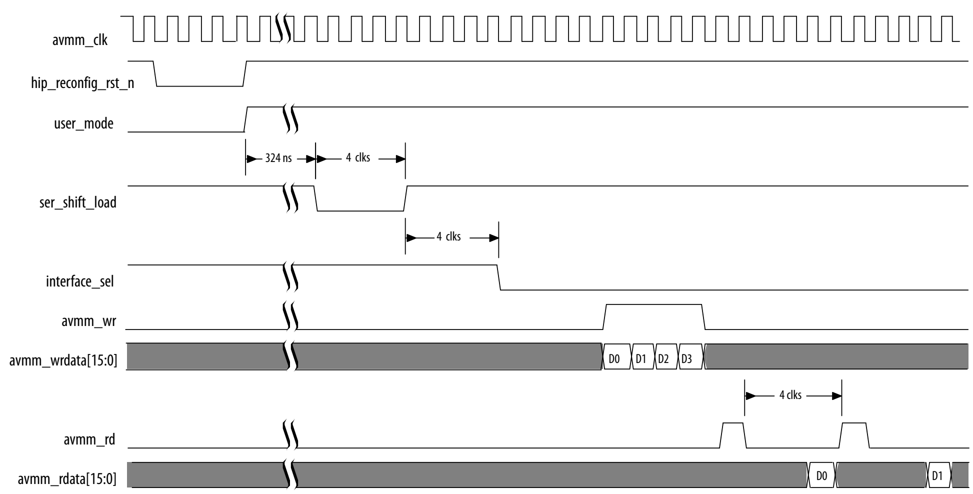
5.5. Hard IP Reconfiguration Interface

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 50. 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.
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 28. 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.

5.6. 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.

5.6.1. 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.

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 51. 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 34
  Illustrates device packages with 1-4 PCIe hard IP blocks
- Hard IP Block Placement In Intel Cyclone 10 GX Devices on page 33
- Pin-out Files for Intel Devices

5.6.2. 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. Note that Intel Arria 10 and Intel Cyclone 10 GX devices do not support the Gen3 PIPE interface. Simulation is faster using the PIPE interface because the PIPE simulation bypasses the SERDES model. By default, the PIPE interface 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 Embedded Logic Analyzer. These signals are not top-level signals of the Hard IP. They are listed here to assist in debugging link training issues.

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.

Table 52. PIPE Interface Signals

In the following table, signals that include lane number 0 also exist for lanes 1-7. These signals are for simulation only. For Quartus Prime software compilation, these pipe signals can be left floating. In Platform Designer, the signals that are part of the PIPE interface have the prefix, *hip_pipe*. The signals which are included to simulate the PIPE interface have the prefix, *hip_pipe_sim_pipe*

Signal	Direction	Description
`currentcoeff0[17:0]`	Output	For Gen3, indicates the coefficients to be used by the transmitter. The 18 bits specify the following coefficients: • [5:0]: C-1 • [11:6]: C0 • [17:12]: C+1
`currentrxpreset0[2:0]`	Output	For Gen3 designs, specifies the current preset.
`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 the in 128 us window for Gen1 or Gen2 • 3'b101: Absence of TS1/TS2 Ordered Set in a 1280 UI interval for Gen1 or Gen2 • 3'b110: Absence of Electrical Idle Exit in 2000 UI interval for Gen1 and 16000 UI interval for Gen2 • 3'b111: Absence of Electrical idle exit in 128 us window for Gen1
`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).
`rate[1:0]`	Output	Controls the link signaling rate. The following encodings are defined: • 2'b00: Gen1 • 2'b01: Gen2 • 2'b10: Gen3 • 2'b11: Reserved
`rxblkst0`	Input	For Gen3 operation, indicates the start of a block in the receive direction.
`rxdata0[31:0]`	Input	Receive data. This bus receives data on lane <*n*>.
`rxdatak0[3:0]`	Input	Data/Control bits for the symbols of receive data. 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.
`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 and error codes for the receive data stream and receiver detection.

continued...

Signal	Direction	Description
`rxvalid0`	Input	Receive valid <*n*>. This symbol indicates symbol lock and valid data on `rxdata<n>` and `rxdatak <n>`.
`sim_pipe_ltssmstate0[4:0]`	Input and Output	LTSSM state: The LTSSM state machine encoding defines the following states: • 5'b00000: Detect.Quiet • 5'b 00001: Detect.Active • 5'b00010: Polling.Active • 5'b 00011: Polling.Compliance • 5'b 00100: Polling.Configuration • 5'b00101: Polling.Speed • 5'b00110: Config.LinkwidthsStart • 5'b 00111: Config.Linkaccept • 5'b 01000: Config.Lanenumaccept • 5'b01001: Config.Lanenumwait • 5'b01010: Config.Complete • 5'b 01011: Config.Idle • 5'b01100: Recovery.Rcvlock • 5'b01101: Recovery.Rcvconfig • 5'b01110: Recovery.Idle • 5'b 01111: L0 • 5'b10000: Disable • 5'b10001: Loopback.Entry • 5'b10010: Loopback.Active • 5'b10011: Loopback.Exit • 5'b10100: Hot.Reset • 5'b10101: 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
`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_rate[1:0]`	Input	Specifies the data rate. 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)
`txblkst`		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).
`txdata0[31:0]`	Output	Transmit data. 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 `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.
`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

continued...

Signal	Direction	Description
txdeemph0	Output	Transmit de-emphasis selection. The value for this signal is set 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.
txmargin0[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.
txswing0	Output	When asserted, indicates full swing for the transmitter voltage. When deasserted indicates half swing.
txsynchd0[1:0]	Output	For Gen3 operation, specifies the block type. The following encodings are defined: • 2'b01: Ordered Set Block • 2'b10: Data Block

5.7. Test Signals

Table 53. 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.
currentspeed[1:0]	Output	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

5.8. 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 54. 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]` • `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`

6. Registers

6.1. Correspondence between Configuration Space Registers and the PCIe Specification

Table 55. 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 The Type 1 Configuration Space is not available for the Avalon-MM with DMA interface
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

continued...

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
Overview of Configuration Space Register Fields		
0x000	Device ID, Vendor ID	Type 0 Configuration Space Header Type 1 Configuration Space Header The Type 1 Configuration Space is not available for the Avalon-MM with DMA interface
0x004	Status, Command	Type 0 Configuration Space Header Type 1 Configuration Space Header The Type 1 Configuration Space is not available for the Avalon-MM with DMA interface
0x008	Class Code, Revision ID	Type 0 Configuration Space Header Type 1 Configuration Space Header The Type 1 Configuration Space is not available for the Avalon-MM with DMA interface
0x00C	BIST, Header Type, Primary Latency Timer, Cache Line Size	Type 0 Configuration Space Header Type 1 Configuration Space Header The Type 1 Configuration Space is not available for the Avalon-MM with DMA interface
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

continued...

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
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
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 29. 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 68 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 30. 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 31. 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 32. 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 33. Power Management Capability Structure - Byte Address Offsets and Layout

	31	24 23	16 15	8 7	0
0x078	Capabilities Register		Next Cap Ptr	Capability ID	
0x07C	Data	PM Control/Status Bridge Extensions	Power Management Status and Control		

Figure 34. 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 35. 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.

Note: The Avalon-MM with DMA interface does not support Root Ports.

Offset	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

6.5. Intel-Defined VSEC Registers

Figure 36. 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 56. 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 57. 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 58. 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 59. 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 60. 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

6.5.1. CvP Registers

Table 61. 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...

Send Feedback

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 62. 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 63. 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 64. 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

6.6. Advanced Error Reporting Capability

6.6.1. Uncorrectable Internal Error Mask Register

Table 65. 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

continued...

Bits	Register Description	Reset Value	Access
[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

6.6.2. Uncorrectable Internal Error Status Register

Table 66. 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

6.6.3. Correctable Internal Error Mask Register

Table 67. 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
[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

6.6.4. Correctable Internal Error Status Register

Table 68. 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

6.7. DMA Descriptor Controller Registers

The DMA Descriptor Controller manages Read and Write DMA operations. The DMA Descriptor Controller is available for use with Endpoint variations. The Descriptor Controller supports up to 128 descriptors each for Read and Write Data Movers. Read and Write are from the perspective of the FPGA. A read is from PCIe address space to the FPGA Avalon-MM address space. A write is to PCIe address space from the FPGA Avalon-MM space.

You program the Descriptor Controller internal registers with the location and size of the descriptor table residing in the PCIe address space. The DMA Descriptor Controller instructs the Read Data Mover to copy the table to its own internal FIFO. When the

DMA Descriptor Controller is instantiated as a separate component, it drives table entries on the RdDmaRxData_i[159:0] and WrDmaRxData_i[159:0] buses. When the DMA Descriptor Controller is embedded inside the Avalon-MM DMA bridge, it drives this information on internal buses. .

The Read Data Mover transfers data from the PCIe address space to Avalon-MM address space. It issues memory read TLPs on the PCIe link. It writes the data returned to a location in the Avalon-MM address space. The source address is the address for the data in the PCIe address space. The destination address is in the Avalon-MM address space.

The Write Data Mover reads data from the Avalon-MM address space and writes to the PCIe address space. It issues memory write TLPs on the PCIe link. The source address is in the Avalon-MM address space. The destination address is in the PCIe address space.

The DMA Descriptor Controller records the completion status for read and write descriptors in separate status tables. Each table has 128 consecutive DWORD entries that correspond to the 128 descriptors. The actual descriptors are stored immediately after the status entries at offset 0x200 from the values programmed into the RC Read Descriptor Base and RC Write Descriptor Base registers. The status and descriptor table must be located on a 32-byte boundary in Root Complex memory.

The Descriptor Controller writes a 1 to the done bit of the status DWORD to indicate successful completion. The Descriptor Controller also sends an MSI interrupt for the final descriptor. After receiving this MSI, host software can poll the done bit to determine status. The status table precedes the descriptor table in memory. The Descriptor Controller does not write the done bit nor send an MSI as each descriptor completes. It only writes the done bit or sends an MSI for the descriptor whose ID is stored in the RD_DMA_LAST PTR or WR_DMA_LAST_PTR registers. The Descriptor Controller supports out-of-order completions. Consequently, it is possible for the done bit to be set before all descriptors have completed.

Note: The following example clarifies this programming model. If 128 descriptors are specified and all of them execute, then descriptor ID 127 is written to the RD_DMA_LAST_PTR or WR_DMA_LAST_PTR register to start the DMA. The DMA Descriptor Controller only writes the done bit when descriptor 127 completes. To get intermediate status updates, host software should write multiple IDs into the last pointer register. For example, to get an intermediate status update when half of the 128 read descriptors have completed, host software should complete the following sequence:

1. Program the RD_DMA_LAST_PTR = 63.
2. Program the RD_DMA_LAST_PTR = 127.
3. Poll the status DWORD for read descriptor 63.
4. Poll the status DWORD for read descriptor 127.

Many commercial system Root Ports return out-of-order Read Completions based on optimized accesses to host memory channels. Consequently, the done status stored for descriptor <*n*> does not necessarily mean that descriptors <*n-1*> and <*n -2*> have also completed. You must request the completion status for every descriptor by writing the descriptor ID for every descriptor to RD_DMA_LAST_PTR or WR_DMA_LAST_PTR.

Note: Because the DMA Descriptor Controller uses FIFOs to store descriptor table entries, you cannot reprogram the DMA Descriptor Controller once it begins the transfers specified in the descriptor table.

6.7.1. Read DMA Descriptor Controller Registers

The following table describes the registers in the internal DMA Descriptor Controller. When the DMA Descriptor Controller is externally instantiated, these registers are accessed through a BAR. The offsets must be added to the base address for the read controller. When the Descriptor Controller is internally instantiated these registers are accessed through BAR0. The read controller is at offset 0x0000.

Address Offset	Register	Access	Description
0x0000	`RC Read Status and Descriptor Base (Low)`	R/W	Specifies the lower 32-bits of the base address of the read status and descriptor table in the Root Complex memory. This address must be on a 32-byte boundary. Software must program this register after programming the upper 32 bits at offset 0x4. To change the `RC Read Status and Descriptor Base (Low)`base address, all descriptors specified by the RD_TABLE_SIZE must be exhausted.
0x0004	`RC Read Status and Descriptor Base (High)`	R/W	Specifies the upper 32-bits of the base address of the read status and descriptor table in the Root Complex memory. Software must program this register before programming the lower 32 bits of this register.
0x0008	`EP Read Descriptor FIFO Base (Low)`	RW	Specifies the lower 32 bits of the base address of the read descriptor FIFO in Endpoint memory. The Read DMA fetches the descriptors from Root Complex memory. The address must be the Avalon-MM address of the Descriptor Controller's Read Descriptor Table Avalon-MM Slave Port as seen by the Read DMA Avalon-MM Master Port. You must program this register after programming the upper 32 bits at offset 0xC.
0x000C	`EP Read Descriptor FIFO Base (High)`	RW	Specifies the upper 32 bits of the base address of the read descriptor table in Endpoint Avalon-MM memory. The Read DMA fetches the descriptors from Root Complex memory and writes the descriptors to the FIFO at this location. This must be the Avalon-MM address of the descriptor controller's Read Descriptor Table Avalon-MM Slave Port as seen by the Read DMA Avalon-MM Master Port. You must program this register before programming the lower 32 bits of this register.
0x0010	`RD_DMA_LAST_PTR`	RW	When read, returns the ID of the last descriptor requested. If no DMA request is outstanding or the DMA is in reset, returns a value 0xFF. When written, specifies the ID of the last descriptor requested. The difference between the value read and the value written is the number of descriptors to be processed. For example, if the value reads 4, the last descriptor requested is 4. To specify 5 more descriptors, software should write a 9 into the RD_DMA_LAST_PTR register. The DMA executes 5 more descriptors. To have the read DMA record the done bit of every descriptor, program this register to transfer one descriptor at a time.

continued...

Send Feedback

Address Offset	Register	Access	Description
			The descriptor ID loops back to 0 after reaching RD_TABLE_SIZE. For example, if the RD_DMA_LAST_PTR value read is 126 and you want to execute three more descriptors, software must write 127, and then 1 into the RD_DMA_LAST_PTRregister.
0x0014	RD_TABLE_SIZE	RW	Specifies the size of the Read descriptor table. Set to the number of descriptors - 1. By default, RD_TABLE_SIZE is set to 127.This value specifies the last `Descriptor ID`. To change the `RC Read Status and Descriptor Base (Low)`base address, all descriptors specified by the RD_TABLE_SIZE must be exhausted.
0x0018	RD_CONTROL	RW	[31:1] Reserved. [0]`Done`. When set, the Descriptor Controller writes the `Done` bit for each descriptor in the status table. When not set the Descriptor Controller writes the Done for the final descriptor, as specified by RD_DMA_LAST_PTR. In both cases, the Descriptor Controller sends a MSI to the host after the completion of the last descriptor along with the status update for the last descriptor.

6.7.2. Write DMA Descriptor Controller Registers

The following table describes the registers in the internal DMA Descriptor Controller. When the DMA Descriptor Controller is externally instantiated, these registers are accessed through a BAR. The offsets must be added to the base address for the write controller. When the Descriptor Controller is internally instantiated these registers are accessed through BAR0. The write controller is at offset 0x0100.

Address Offset	Register	Access	Description
0x0100	`RC Write Status and Descriptor Base (Low)`	R/W	Specifies the lower 32-bits of the base address of the write status and descriptor table in the Root Complex memory. This address must be on a 32-byte boundary. Software must program this register after programming the upper 32-bit register at offset 0x104. To change the RC `Write Status and Descriptor Base (Low)` base address, all descriptors specified by the WR_TABLE_SIZE must be exhausted.
0x0104	`RC Write Status and Descriptor Base (High)`	R/W	Specifies the upper 32-bits of the base address of the write status and descriptor table in the Root Complex memory. Software must program this register before programming the lower 32-bit register at offset 0x100.
0x0108	`EP Write Status and Descriptor FIFO Base (Low)`	RW	Specifies the lower 32 bits of the base address of the write descriptor table in Endpoint memory. The Write Descriptor Controller requests descriptors from Root Complex memory and writes the descriptors to this location. The address is the Avalon-MM address of the Descriptor Controller's Write Descriptor Table Avalon-MM Slave Port as seen by the Read DMA Avalon-MM Master Port. Software must program this register after programming the upper 32-bit register at offset 0x10C.

continued...

Address Offset	Register	Access	Description
0x010C	EP Write Status and Descriptor FIFO Base (High)	RW	Specifies the upper 32 bits of the base address of the write descriptor table in Endpoint memory. The read DMA fetches the table from Root Complex memory and writes the table to this location. Software must program this register before programming the lower 32-bit register at offset 0x108.
0x0110	WR_DMA_LAST_PTR	RW	When read, returns the ID of the last descriptor requested. If no DMA request is outstanding or the DMA is in reset, returns a value 0xFF. When written, specifies the ID of the last descriptor requested. The difference between the value read and the value written is the number of descriptors to be processed. For example, if the value reads 4, the last descriptor requested is 4. To specify 5 more descriptors, software should write a 9 into the RD_DMA_LAST_PTR register. The DMA executes 5 more descriptors. To have the write DMA record the done bit of every descriptor, program this register to transfer one descriptor at a time. The Descriptor ID loops back to 0 after reaching WR_TABLE_SIZE. For example, if the WR_DMA_LAST_PTR value read is 126 and you want to execute three more descriptors, software must write 127, and then 1 into the WR_DMA_LAST_PTR register.
0x0114	WR_TABLE_SIZE	RW	Specifies the size of the Read descriptor table. Set to the number of descriptors - 1. By default, WR_TABLE_SIZE is set to 127. This value specifies the last Descriptor ID. To change the RC Write Status and Descriptor Base (Low) base address, all descriptors specified by the WR_TABLE_SIZE must be exhausted.
0x0118	WR_CONTROL	RW	[31:1] Reserved. [0]Done. When set, the Descriptor Controller writes the Done bit for each descriptor in the status table. The Descriptor Controller sends a single MSI interrupt after the final descriptor completes. When not set the Descriptor Controller generates Done for the final descriptor, as specified by WR_DMA_LAST_PTR. In both cases, the Descriptor Controller sends an MSI to the host after the completion of the last descriptor along with the status update for the last descriptor.

6.7.3. Read DMA and Write DMA Descriptor Format

Read and write descriptors are stored in separate descriptor tables in PCIe system memory. Each table can store up to 128 descriptors. Each descriptor is 8 DWORDs, or 32 bytes. The Read DMA and Write DMA descriptor tables start at a 0x200 byte offset from the addresses programmed into the Read Status and Descriptor Base and Write Status and Descriptor Base address registers.

Programming RD_DMA_LAST_PTR or WR_DMA_LAST_PTR registers triggers the Read or Write Descriptor Controller descriptor table fetch process. Consequently, writing these registers must be the last step in setting up DMA transfers.

Note: Because the DMA Descriptor Controller uses FIFOs to store descriptor table entries, you cannot reprogram the DMA Descriptor Controller once it begins the transfers specified in the descriptor table.

Table 69. Read Descriptor Format

You must also use this format for the Read and Write Data Movers on their Avalon-ST when you use your own DMA Controller.

Address Offset	Register Name	Description
0x00	RD_LOW_SRC_ADDR	Lower DWORD of the read DMA source address. Specifies the address in PCIe system memory from which the Read Data Mover fetches data.
0x04	RD_HIGH_SRC_ADDR	Upper DWORD of the read DMA source address. Specifies the address in PCIe system memory from which the Read Data Mover fetches data.
0x08	RD_CTRL_LOW_DEST_ADDR	Lower DWORD of the read DMA destination address. Specifies the address in the Avalon-MM domain to which the Read Data Mover writes data.
0x0C	RD_CTRL_HIGH_DEST_ADDR	Upper DWORD of the read DMA destination address. Specifies the address in the Avalon-MM domain to which the Read Data Mover writes data.
0x10	CONTROL	Specifies the following information: • [31:25] Reserved must be 0. • [24:18] ID. Specifies the Descriptor ID. Descriptor ID 0 is at the beginning of the table. For descriptor tables of the maximum size, Descriptor ID 127 is at the end of the table. • [17:0] SIZE. The transfer size in DWORDs. Must be non-zero. The maximum transfer size is (1 MB - 4 bytes). If the specified transfer size is less than the maximum, the transfer size is the actual size entered.
0x14 - 0x1C	Reserved	N/A

Table 70. Write Descriptor Format

Address Offset	Register Name	Description
0x00	WR_LOW_SRC_ADDR	Lower DWORD of the write DMA source address. Specifies the address in the AvalonMM domain from which the Write Data Mover fetches data.
0x04	WR_HIGH_SRC_ADDR	Upper DWORD of the write DMA source address. Specifies the address in the AvalonMM domain from which the Write Data Mover fetches data.
0x08	WR_CTRL_LOW_DEST_ADDR	Lower DWORD of the Write Data Mover destination address. Specifies the address in PCIe system memory to which the Write DMA writes data.

continued...

Address Offset	Register Name	Description
0x0C	WR_CTRL_HIGH_DEST_ADDR	Upper DWORD of the write DMA destination address. Specifies the address in PCIe system memory to which the Write Data Mover writes data.
0x10	CONTROL	Specifies the following information: • [31:25]: Reserved must be 0. • [24:18]:ID: Specifies the Descriptor ID. Descriptor ID 0 is at the beginning of the table. Descriptor ID is at the end of the table. • [17:0] :SIZE: The transfer size in DWORDs. Must be non-zero. The maximum transfer size is (1 MB - 4 bytes). If the specified transfer size is less than the maximum, the transfer size is the actual size entered.
0x14 - 0x1C	Reserved	N/A

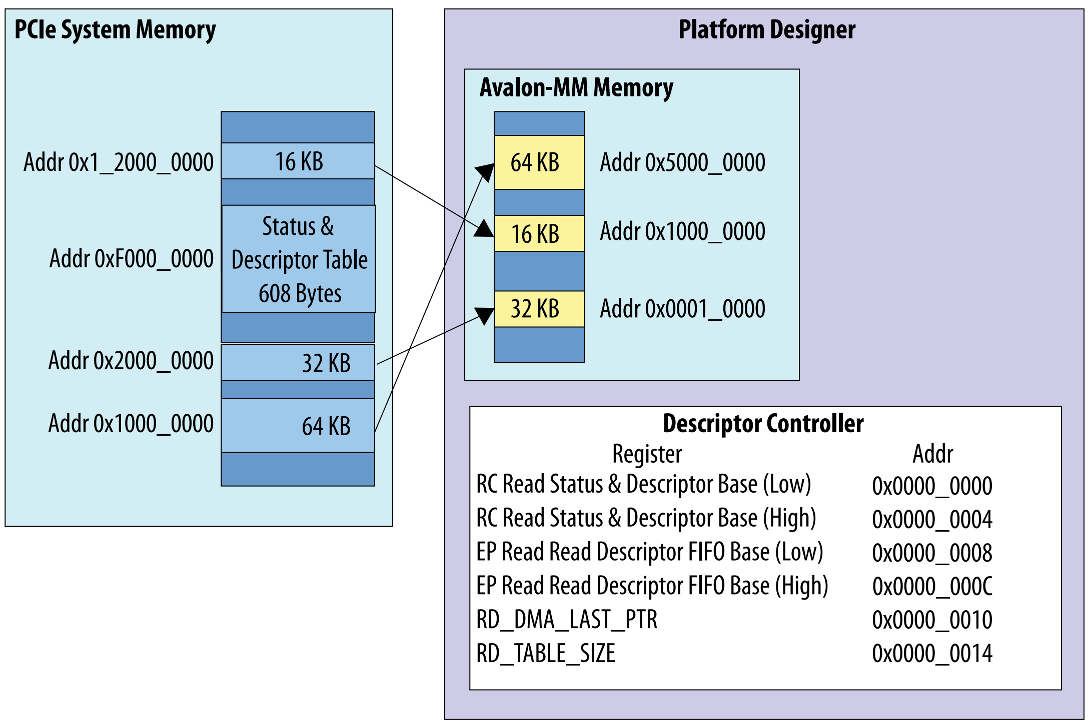
6.7.4. Read DMA Example

This example moves three data blocks from the system memory to the Avalon-MM address space. Host software running on an embedded CPU allocates the memory and creates the descriptor table in system memory.

This example uses the addresses in the Platform Designer design example, `ep_g3x8_avmm256_integrated.qsys`, available in the <install_dir>/ ip/altera/altera_pcie/altera_pcie_<dev>_ed/example_design/<dev> directory.

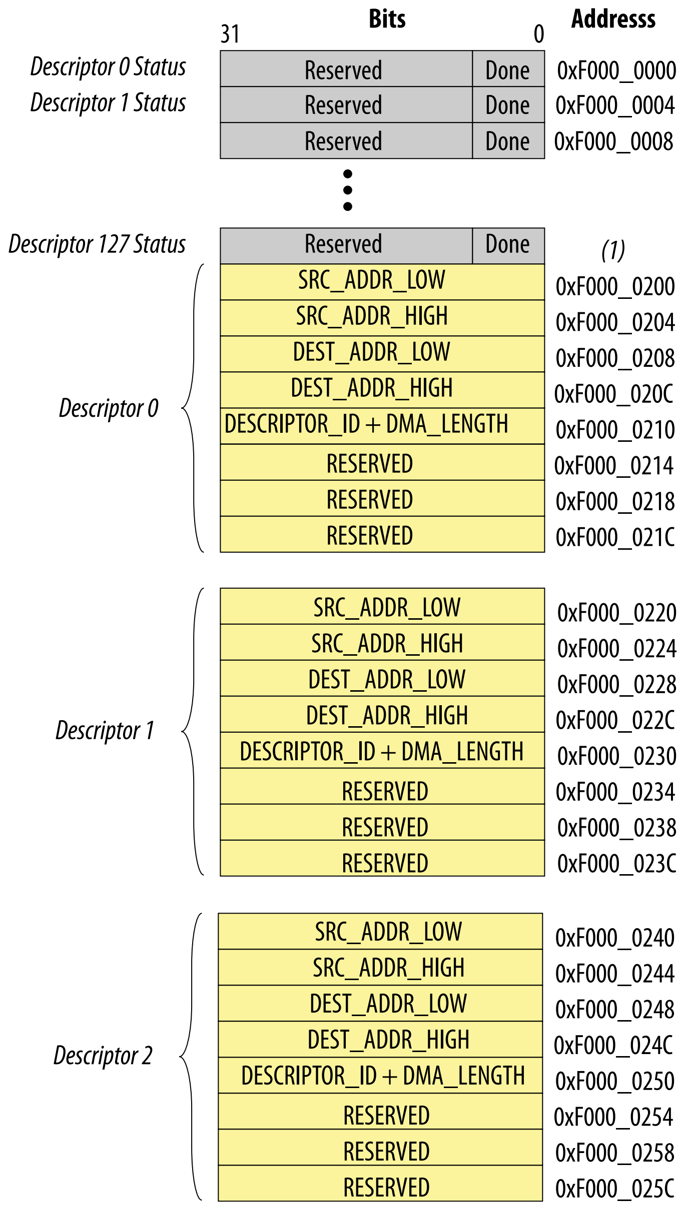
The following figures illustrate the location and size of the data blocks in the PCIe and Avalon-MM address spaces and the descriptor table format. In this example, the value of RD_TABLE_SIZE is 127.

Figure 37. Data Blocks to Transfer from PCIe to Avalon-MM Address Space Using Read DMA



Assume the descriptor table includes 128 entries. The status table precedes a variable number of descriptors in memory. The Read and Write Status and Descriptor Tables are at the address specified in the `RC Read Descriptor Base Register` and `RC Write Descriptor Base Register`, respectively.

Figure 38. Descriptor Table Format



Note:
1. Software automatically adds 0x200 to the base address of the status table to determine the address of the first read descriptor.

1. Calculate the memory allocation required:
   a. Each entry in the status table is 4 bytes. The 128 entries require 512 bytes of memory.
   b. Each descriptor is 32 bytes. The three descriptors require 96 bytes of memory.

   The total memory allocation for the status and descriptor tables is 608 bytes.
2. Allocate 608 bytes of memory in the PCI Express address space.

The start address of the allocated memory in this example is 0xF000_0000. Program this address into the Root Complex Read Status and Descriptor Base Address Registers.

3. Create the descriptor table in the PCI Express address space. Because the status table is stored before the descriptors, the first descriptor is stored at 0xF000_0000 + 0x200 = 0xF000_0200.
    a. Program 0x0000_0000 into the source address 0xF000_0204 in descriptor 0.

       This is the upper 32 bits of the source address.
    b. Program 0x1000_0000 into the source address 0xF000_0200 in descriptor 0.

       This is the lower 32 bits of the source address.
    c. Program 0x0000_0000 into the destination address 0xF000_020C in descriptor 0.

       This is the upper 32 bits of the destination address.
    d. Program 0x5000_0000 into the destination address 0xF000_0208 in descriptor 0.

       This is the lower 32 bits of the destination address.

       These four steps program the Avalon-MM destination address for the 64 KB block of memory into the Descriptor Table.
    e. Program 0x0000_4000 to 0xF000_0210 to transfer 16K dwords (64 KB), of data for descriptor ID 0.
4. Repeat this procedure for the second data block:
    a. Program 0x0000_0000 to source address 0xF000_0224.
    b. Program 0x2000_0000 to source address 0xF000_0220.
    c. Program 0x0000_0000 to destination address 0xF000_022C.
    d. Program 0x0001_0000 to destination address 0xF000_0228.
    e. Program 0x0004_2000 to 0xF000_0230 to transfer 8K dwords (32 KB) of data for descriptor ID 1.
5. Repeat this procedure for the third data block:
    a. Program 0x0000_0001 to source address 0xF000_0244.
    b. Program 0x2000_0000 to source address 0xF000_0240.
    c. Program 0x0000_0000 to destination address 0xF000_024C.
    d. Program 0x1000_0000 to destination address 0xF000_0248.
    e. Program 0x0008_1000 to 0xF000_0250 to transfer 4K dwords (16 KB) of data for descriptor ID 2.

The following figure shows the values in the Descriptor Table after programming completes.

Figure 39. Descriptor Table Format

	Bits (31)	(0)	Address
Descriptor 0 Status	Reserved	Done	0xF000_0000
Descriptor 1 Status	Reserved	Done	0xF000_0004
	Reserved	Done	0xF000_0008
⋮			
Descriptor 127 Status	Reserved	Done	(1)
Descriptor 0	1000_0000		0xF000_0200
	0000_0000		0xF000_0204
	5000_0000		0xF000_0208
	0000_0000		0xF000_020C
	0000_4000		0xF000_0210
	RESERVED		0xF000_0214
	RESERVED		0xF000_0218
	RESERVED		0xF000_021C
Descriptor 1	2000_0000		0xF000_0220
	0000_0000		0xF000_0224
	0001_0000		0xF000_0228
	0000_0000		0xF000_022C
	0004_2000		0xF000_0230
	RESERVED		0xF000_0234
	RESERVED		0xF000_0238
	RESERVED		0xF000_023C
Descriptor 2	2000_0000		0xF000_0240
	0000_0001		0xF000_0244
	1000_0000		0xF000_0248
	0000_0000		0xF000_024C
	0008_1000		0xF000_0250
	RESERVED		0xF000_0254
	RESERVED		0xF000_0258
	RESERVED		0xF000_025C

Note:
1. The DMA Descriptor Controller automatically adds 0x200 to the base address of the staus table to determine the address of the first read descriptor.

6. Program the DMA Descriptor Controller with the address of the status and descriptor table in the PCI Express system memory address space. When the DMA Descriptor Controller is internal, these registers are accessed through combined BAR0 and BAR1 because this example uses 64-bit addresses. The DMA read control registers start at offset 0x0000. The Write DMA control registers start at offset 0x0100

a. Program 0x0000_0000 to offset 0x0000_0004.

This is the upper 32 bits of the PCIe system memory where the status and descriptor table is stored.

b. Program 0xF000_0000 to offset 0x0000_0000.

This is the lower 32 bits of the address in PCIe memory that stores the status and descriptor tables. The Read DMA automatically adds an offset of 0x200 to this value to start the copy of the descriptors which follow the status table in memory.

7. Program the DMA Descriptor Controller with the on-chip FIFO address. This is the address to which the Descriptor Controller copies the status and descriptor table.

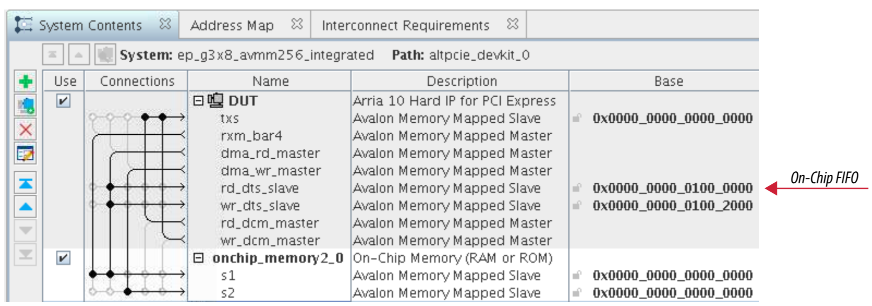
a. Program 0x0000_0000 to offset 0x0000_000C

This is the upper 32 bits of the on-chip FIFO address in the Avalon-MM address domain.

b. Program 0x0100_0000 to offset 0x0000_0008.

This is the lower 32 bits of the on-chip FIFO address. This is address of the internal on-chip FIFO that is a part of the Descriptor Controller as seen by the RX Master.

Figure 40. Address of the On-Chip FIFO



8. Program the Descriptor Controller RD_DMA_LAST_PTR register.

This step starts the DMA. It also specifies the status dword to be updated when the three descriptors complete.

— To update a single done bit for the final descriptor, program 0x2 to offset 0x0000_0010. The Descriptor Controller processes all three descriptors and writes the done bit to 0xF000_0008 of the status table.

— To update the done bits for all three descriptors, program address 0x0000_0010 RD_DMA_LAST_PTR three times with the values 0, 1, and 2. The Descriptor Controller sets the done bits for addresses 0xF000_0000, 0xF000_0004, and 0xF000_0008. If the system returns Read Completions out-of-order, the Descriptor Controller may complete descriptors out of order. In such systems, you must use this method of requesting done status for each descriptor. Software must check for done status for every descriptor.

Related Information

Understanding the Internal DMA Descriptor Controller on page 114

6.7.5. Software Program for Simultaneous Read and Write DMA

Program the following steps to implement a simultaneous DMA transfer:

1. Allocate PCIe system memory for the Read and Write DMA descriptor tables. If, for example, each table supports up to 128, eight-DWORD descriptors and 128, one-DWORD status entries for a total of 1152 DWORDs. Total memory for the Read and Write DMA descriptor tables is 2304 DWORDs.
2. Allocate PCIe system memory and initialize it with data for the Read Data Mover to read.
3. Allocate PCIe system memory for the Write Data Mover to write.
4. Create all the descriptors for the read DMA descriptor table. Assign the DMA `Descriptor IDs` sequentially, starting with 0 to a maximum of 127. For the read DMA, the source address is the memory space allocated in Step 2. The destination address is the Avalon-MM address that the Read Data Mover module writes. Specify the DMA length in DWORDs. Each descriptor transfers contiguous memory. Assuming a base address of 0, for the Read DMA, the following assignments illustrate construction of a read descriptor:
   a. `RD_LOW_SRC_ADDR = 0x0000` (The base address for the read descriptor table in the PCIe system memory.)
   b. RD_HIGH_SRC_ADDR = 0x0004
   c. RD_CTRL_LOW_DEST_ADDR 0x0008
   d. RD_CTRL_HIGH_DEST_ADDR = 0x000C
   e. RD_DMA_LAST_PTR = 0x0010

   Writing the RD_DMA_LAST_PTR register starts operation.
5. For the Write DMA, the source address is the Avalon-MM address that the Write Data Mover module should read. The destination address is the PCIe system memory space allocated in Step 3. Specify the DMA size in DWORDs. Assuming a base address of 0x100, for the Write Data Mover, the following assignments illustrate construction of a write descriptor:
   a. `WD_LOW_SRC_ADDR = 0x0100` (The base address for the write descriptor table in the PCIe system memory.)
   b. WD_HIGH_SRC_ADDR = 0x0104
   c. WD_CTRL_LOW_DEST_ADDR 0x0108
   d. WD_CTRL_HIGH_DEST_ADDR = 0x010C
   e. WD_DMA_LAST_PTR = 0x0110

   Writing the WD_DMA_LAST_PTR register starts operation.
6. To improve throughput, the Read DMA module copies the descriptor table to the Avalon-MM memory before beginning operation. Specify the memory address by writing to the `Descriptor Table Base (Low)` and `(High)` registers.
7. An MSI interrupt is sent for each WD_DMA_LAST_PTR or RD_DMA_LAST_PTR that completes. These completions result in updates to the `done` status bits. Host software can then read status bits to determine which DMA operations are complete.

Note: If the transfer size of the read DMA is greater than the maximum read request size, the Read DMA creates multiple read requests. For example, if maximum read request size is 512 bytes, the Read Data Mover breaks a 4 KB read request into 8 requests with 8 different tags. The Read Completions can come back in any order. The Read Data Mover's Avalon-MM master port writes the data received in the Read Completions to the correct locations in Avalon-MM memory, based on the tags in the same order as it receives the Completions. This order is not necessarily in increasing address order;. The data mover does not include an internal reordering buffer. If system allows out of order read completions, then status for the latest entry is latest only in number, but potentially earlier than other completions chronologically

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

Table 71. Configuration Space Register Descriptions

The optional CRA Avalon-MM slave port provides host access to selected Configuration Space and status registers. These registers are read only. For registers that are less than 32 bits, the upper bits are unused.

Byte Offset	Register	Access	Description
14'h0000	`cfg_dev_ctrl[15:0]`	RO	`cfg_devctrl[15:0]` is device control for the PCI Express capability structure.
14'h0004	`cfg_dev_ctrl2[15:0]`	RO	`cfg_dev2ctrl[15:0]` is device control 2 for the PCI Express capability structure.
14'h0008	`cfg_link_ctrl[15:0]`	RO	`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 even if both devices on the link are capable of a higher data rate.
14'h000C	`cfg_link_ctrl2[15:0]`	RO	`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'h0010	`cfg_prm_cmd[15:0]`	RO	Base/Primary Command register for the PCI Configuration Space.
14'h0014	`cfg_root_ctrl[7:0]`	RO	Root control and status register of the PCI-Express capability. This register is only available in Root Port mode.
14'h0018	`cfg_sec_ctrl[15:0]`	RO	Secondary bus Control and Status register of the PCI-Express capability. This register is only available in Root Port mode.
14'h001C	`cfg_secbus[7:0]`	RO	Secondary bus number. Available in Root Port mode.
14'h0020	`cfg_subbus[7:0]`	RO	Subordinate bus number. Available in Root Port mode.
14'h0024	`cfg_msi_addr_low[31:0]`	RO	`cfg_msi_add[31:0]` is the MSI message address.
14'h0028	`cfg_msi_addr_hi[63:32]`	RO	`cfg_msi_add[63:32]` is the MSI upper message address.
14'h002C	`cfg_io_bas[19:0]`	RO	The IO base register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h0030	`cfg_io_lim[19:0]`	RO	The IO limit register of the Type1 Configuration Space. This register is only available in Root Port mode.

continued...

Byte Offset	Register	Access	Description
14'h0034	`cfg_np_bas[11:0]`	RO	The non-prefetchable memory base register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h0038	`cfg_np_lim[11:0]`	RO	The non-prefetchable memory limit register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h003C	`cfg_pr_bas_low[31:0]`	RO	The lower 32 bits of the prefetchable base register of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h0040	`cfg_pr_bas_hi[43:32]`	RO	The upper 12 bits of the prefetchable base registers of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h0044	`cfg_pr_lim_low[31:0]`	RO	The lower 32 bits of the prefetchable limit registers of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h0048	`cfg_pr_lim_hi[43:32]`	RO	The upper 12 bits of the prefetchable limit registers of the Type1 Configuration Space. This register is only available in Root Port mode.
14'h004C	`cfg_pmcsr[31:0]`	RO	`cfg_pmcsr[31:16]` is Power Management Control and `cfg_pmcsr[15:0]`is the Power Management Status register.
14'h0050	`cfg_msixcsr[15:0]`	RO	MSI-X message control register.
14'h0054	`cfg_msicsr[15:0]`	RO	MSI message control.
14'h0058	`cfg_tcvcmap[23:0]`	RO	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'h005C	`cfg_msi_data[15:0]`	RO	`cfg_msi_data[15:0]` is message data for MSI.
14'h0060	`cfg_busdev[12:0]`	RO	Bus/Device Number captured by or programmed in the Hard IP. The following fields are defined: • cfg_busdev[12:5]: bus number • cfg_busdev[4:0]: device number
14'h0064	`ltssm_reg[4:0]`	RO	Specifies the current LTSSM state. The LTSSM state machine encoding defines the following states: : • 5'b: 00000: Detect.Quiet • 5'b: 00001: Detect.Active • 5'b: 00010: Polling.Active • 5'b: 00011: Polling.Compliance • 5'b: 00100: Polling.Configuration • 5'b: 00101: Polling.Speed • 5'b: 00110: config.Linkwidthstart • 5'b: 00111: Config.Linkaccept

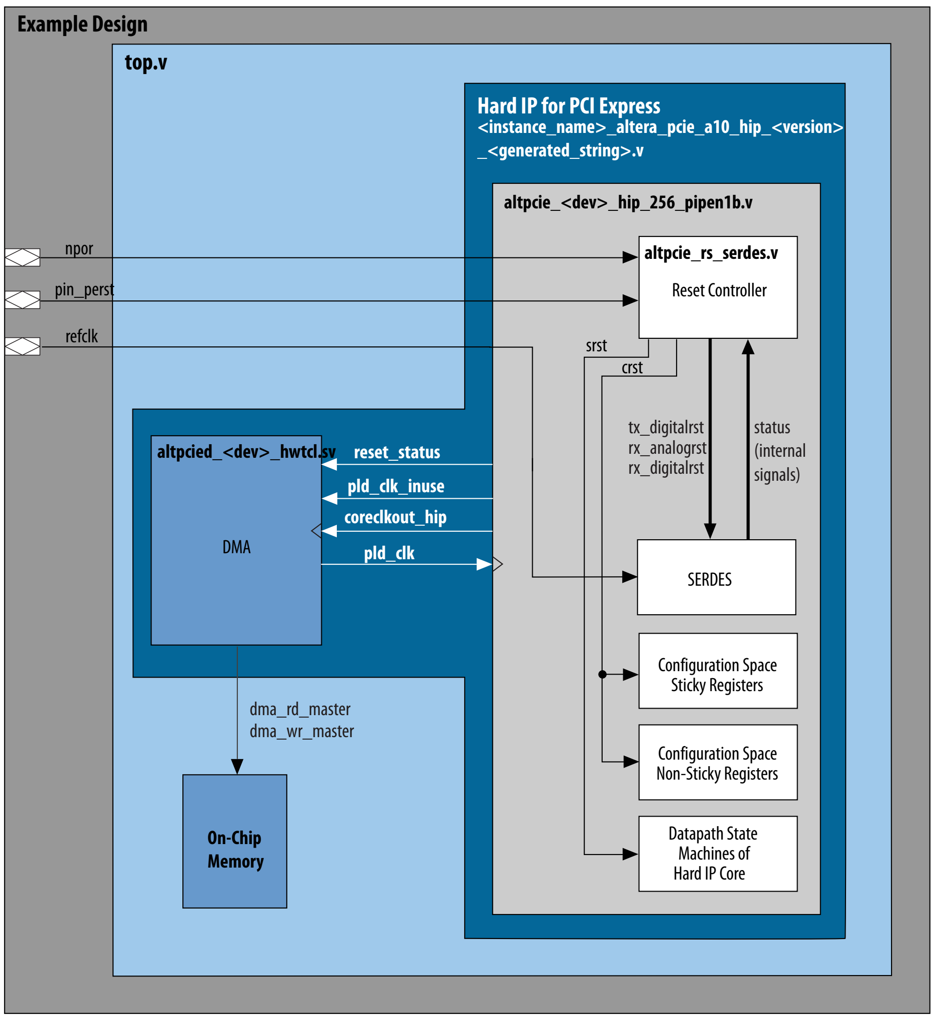
continued...

Byte Offset	Register	Access	Description
			• 5'b: 01000: Config.Lanenumaccept • 5'b: 01001: Config.Lanenumwait • 5'b: 01010: Config.Complete • 5'b: 01011: Config.Idle • 5'b: 01100: Recovery.Rcvlock • 5'b: 01101: Recovery.Rcvconfig • 5'b: 01110: Recovery.Idle • 5'b: 01111: L0 • 5'b: 10000: Disable • 5'b: 10001: Loopback.Entry • 5'b: 10010: Loopback.Active • 5'b: 10011: Loopback.Exit • 5'b: 10100: Hot.Reset • 5'b: 10101: LOs • 5'b: 11001: L2.transmit.Wake • 5'b: 11010: Speed.Recovery • 5'b: 11011: Recovery.Equalization, Phase 0 • 5'b: 11100: Recovery.Equalization, Phase 1 • 5'b: 11101: Recovery.Equalization, Phase 2 • 5'b: 11110: recovery.Equalization, Phase 3
14'h0068	`current_speed_reg[1:0]`	RO	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'h006C	`lane_act_reg[3:0]`	RO	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

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.

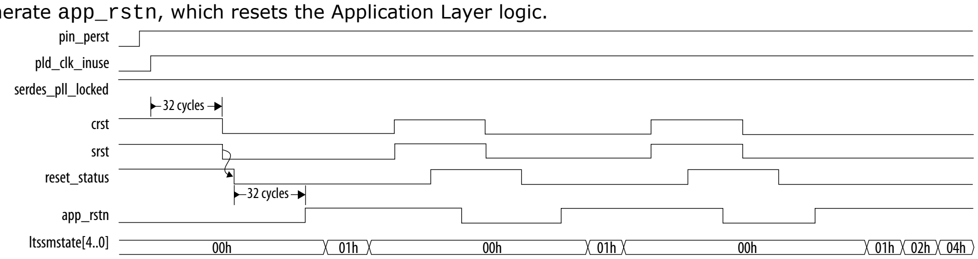
Figure 41. Reset Controller in Intel Arria 10 or Intel Cyclone 10 GX Devices



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

Figure 42. 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.

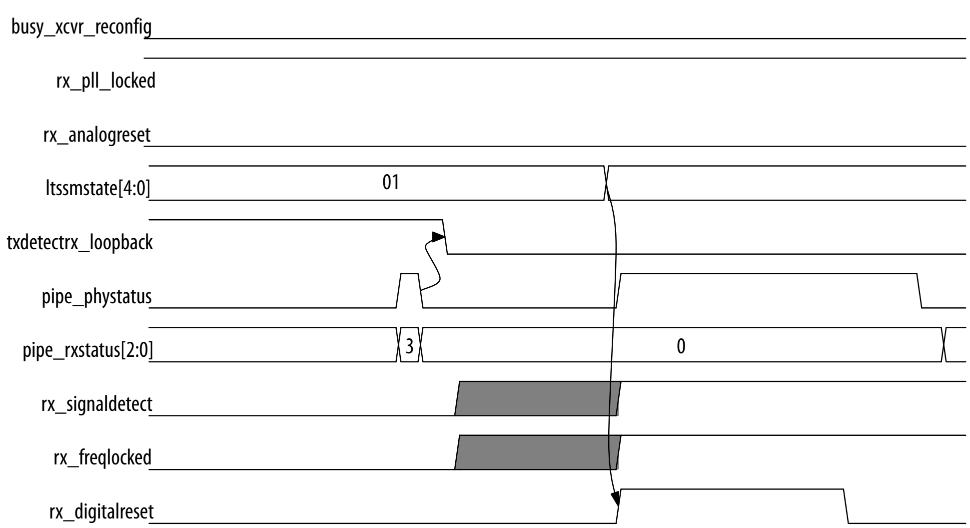


This reset sequence includes the following steps:

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).

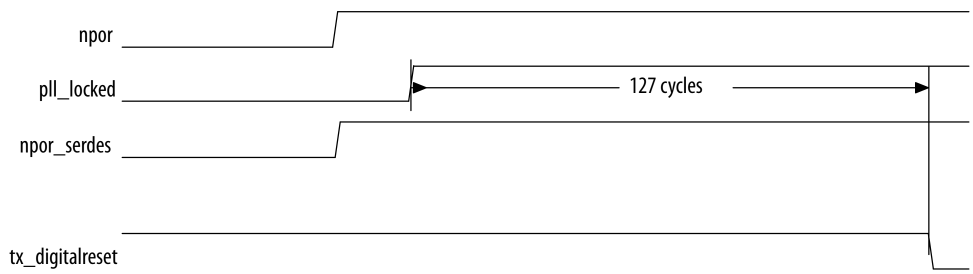
Figure 43. RX Transceiver Reset Sequence



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 44. 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.

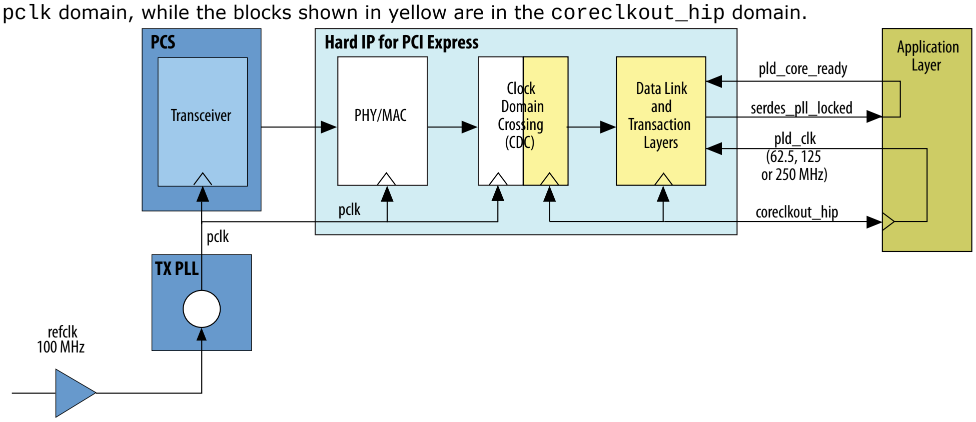
Related Information

PCI Express Base Specification 3.0

7.2.1. Clock Domains

Figure 45. 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 72. 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	Max Link Rate	Avalon Interface Width	coreclkout_hip
×8	Gen1	128	125 MHz
×4	Gen2	128	125 MHz
×8	Gen2	128	250 MHz
×8	Gen2	256	125 MHz
×2	Gen3	128	125 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 73. 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`	125 or 250 MHz	Application and Transaction Layers.
`refclk`	100 MHz	SERDES (transceiver). Dedicated free running input clock to the SERDES block.

8. 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 74. 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

8.1. Physical Layer Errors

Table 75. 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.

8.2. Data Link Layer Errors

Table 76. 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.

8.3. Transaction Layer Errors

Table 77. 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
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. • 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*.

8.4. 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 78. 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

8.5. Uncorrectable and Correctable Error Status Bits

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

Figure 46. Uncorrectable Error Status Register

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

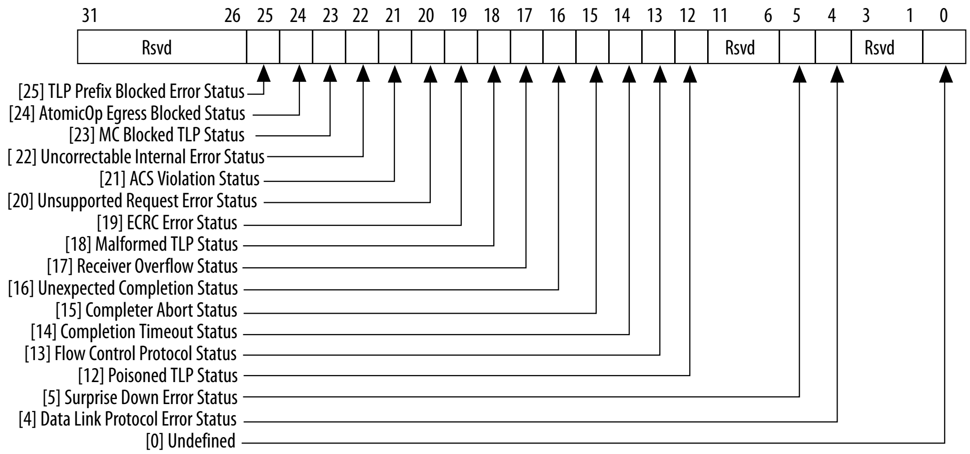
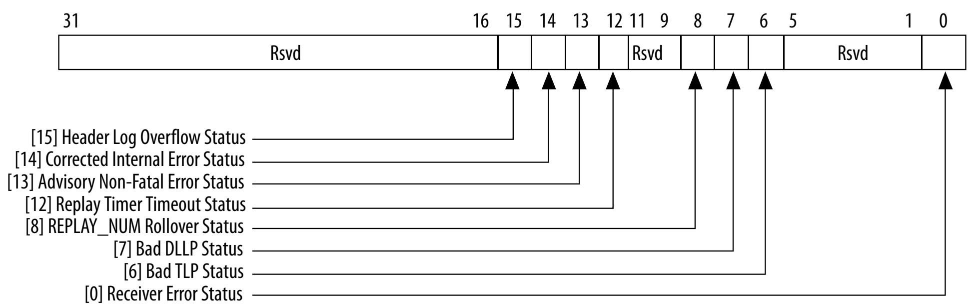


Figure 47. Correctable Error Status Register

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.



9. PCI Express Protocol Stack

The Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM 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 48. Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA for PCI Express

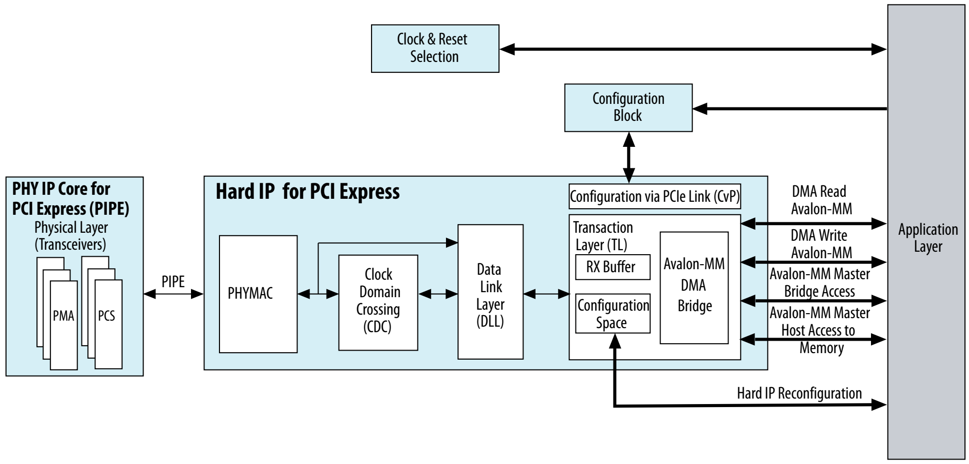


Table 79. Application Layer Clock Frequencies

Lanes	Gen1	Gen2	Gen3
×2	N/A	N/A	125 MHz @ 128 bits
×4	N/A	125 MHz @ 128 bits	250 MHz @ 128 bits or 125 MHz @ 256 bits
×8	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

9.1. Top-Level Interfaces

9.1.1. Avalon-MM DMA Interface

An Avalon-MM interface with DMA connects the Application Layer and the Transaction Layer. This interface includes high-performance, burst capable Read DMA and Write DMA modules. This variant is available for the following Endpoint configurations:

- Gen1 x8
- Gen2 x4, x8
- Gen3 x2, x4, x8

Related Information

Intel Arria 10 or Intel Cyclone 10 GX DMA Avalon-MM DMA Interface to the Application Layer on page 42

9.1.2. 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.

9.1.3. 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

MSI Interrupts for Endpoints on page 61

9.1.4. 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 63

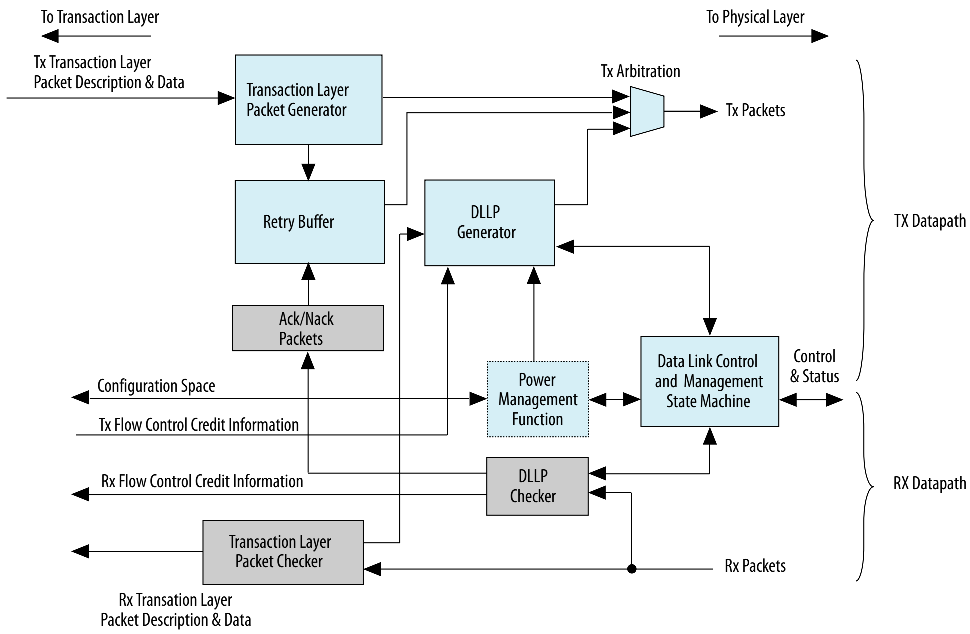
9.2. Data Link Layer

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 49. Data Link Layer



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)

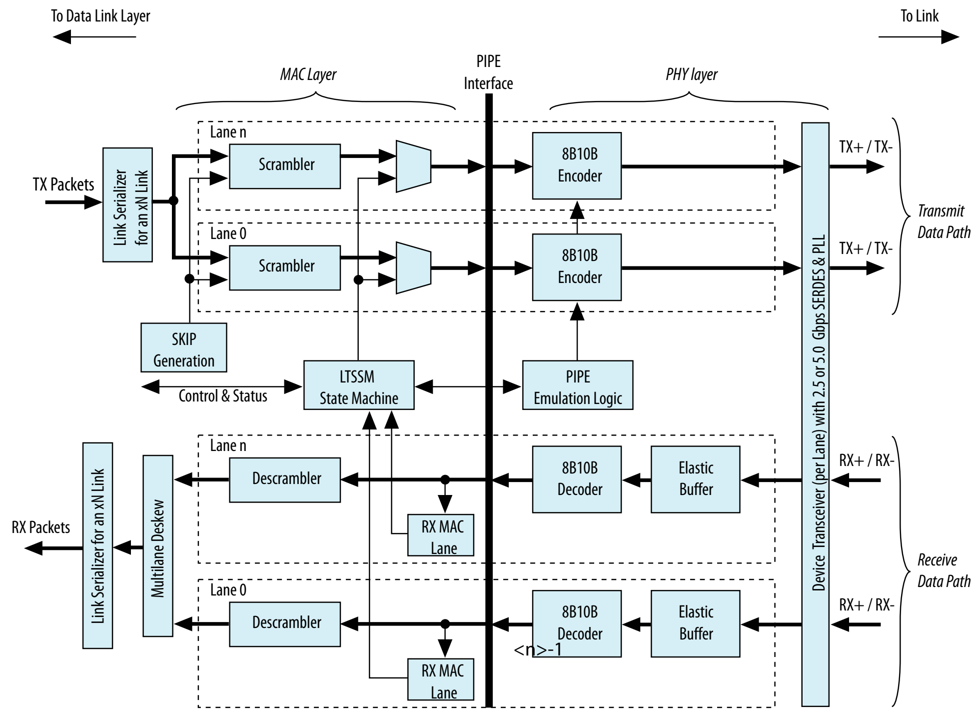
9.3. Physical Layer

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 50. 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.

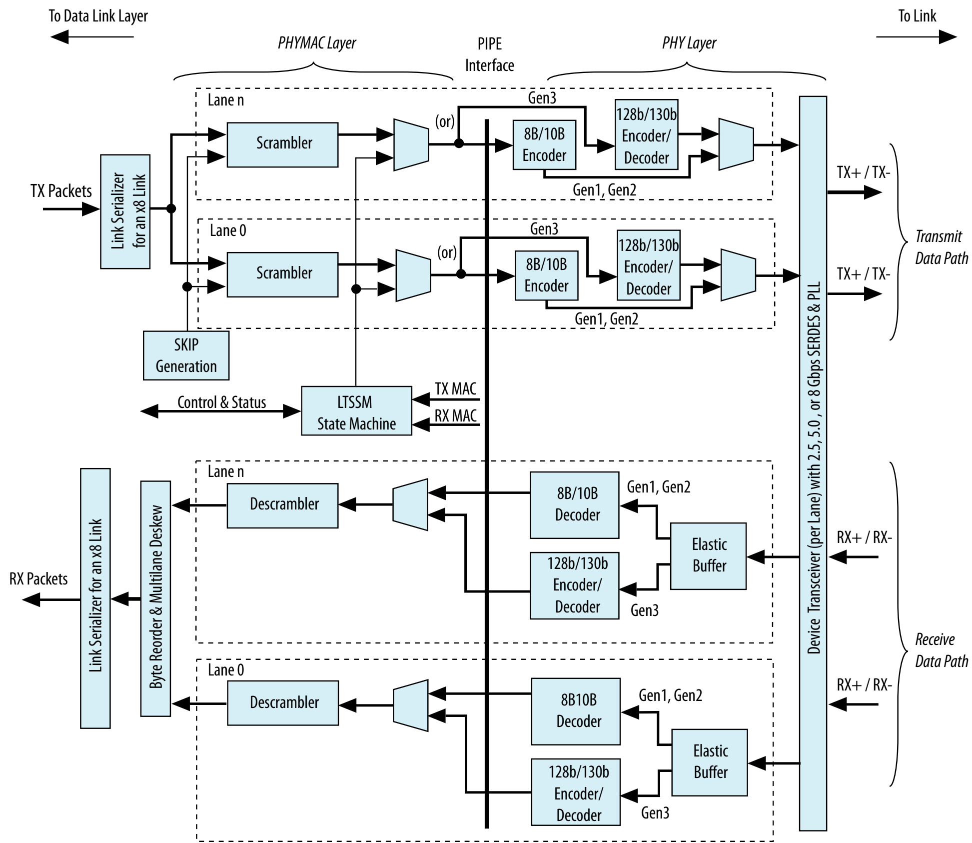
The Physical Layer is subdivided by the PIPE Interface Specification into two layers (bracketed horizontally in above figure):

- 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 51. 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.

10. Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA for PCI Express

The Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA for PCI Express IP Core includes highly efficient Read DMA, Write DMA, and DMA Descriptor Controller modules. The typical throughput for hardware systems using a 256-bit interface to the Application Layer is 6 GB/sec or higher. For hardware systems using a 128-bit interface to the Application Layer the throughput scales proportionately to 3 GB/sec.

Using a 64-byte payload, the maximum theoretical throughput is far less due to the increased proportion of the bandwidth taken by the TLP headers. The throughput for back-to-back TX memory write completions, RX read completions, and simultaneous reads and writes is 2 GB/sec.

Note: A 64-byte packet is the minimum packet size for Ethernet.

Related Information

- Getting Started with the Avalon-MM DMA on page 15
- DMA Descriptor Controller Registers on page 80

10.1. Understanding the Internal DMA Descriptor Controller

When you select **Instantiate internal descriptor controller** in the parameter editor, the Avalon-MM with DMA includes an internal DMA Descriptor Controller to manage read and write DMA operations. The DMA Descriptor Controller includes read and write data movers to perform local memory reads and writes. It supports up to 128 descriptors for read and write DMAs. Host software programs the DMA Descriptor Controller internal registers with the location and size of the descriptor table residing in the PCI Express main memory. The descriptor control logic directs the DMA read logic to copy the entire table to its local FIFOs.

Figure 52. Platform Designer Design Example with the Internal DMA Descriptor Controller

This Platform Designer design example, **ep_g3x8_avmm256_integrated.qsys**, is available in the `<install_dir>/ ip/altera/altera_pcie/altera_pcie_a10_ed/example_design/a10` directory. Refer to *Getting Started with the Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA* for instructions on simulating and compiling this example design. This screen capture filters out some interface types for clarity.

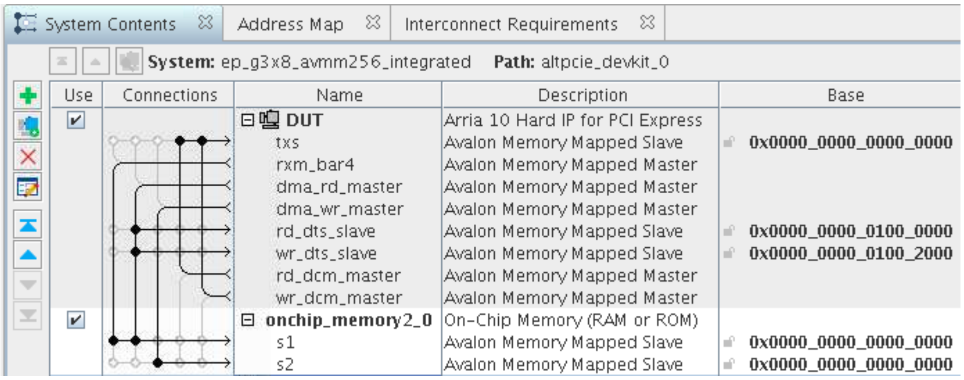
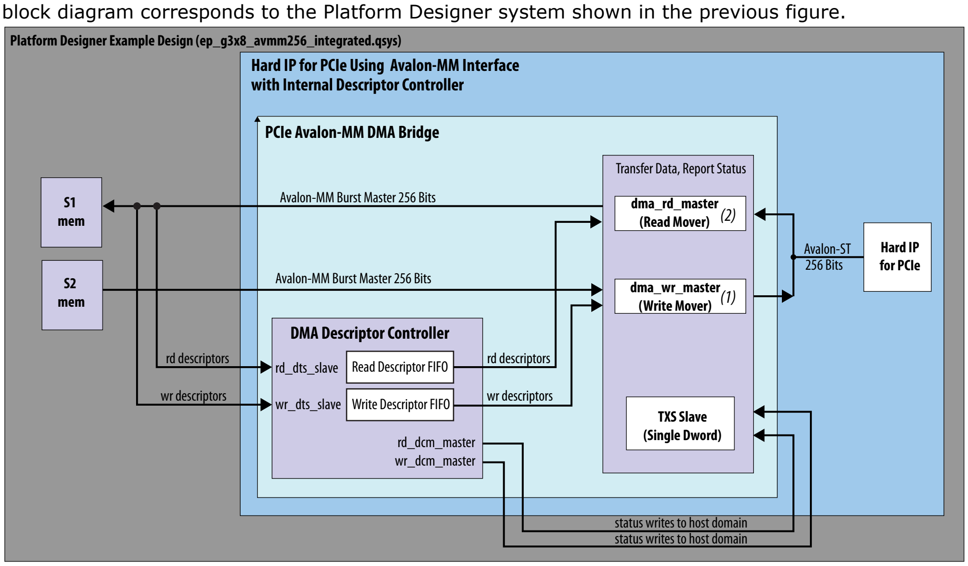


Figure 53. Avalon-MM DMA Block Diagram with the Internal DMA Descriptor Controller

This block diagram corresponds to the Platform Designer system shown in the previous figure.



Notes:

(1) Write Mover transfers data from local domain to the host domain.
(2) Read Mover transfers data from the host domain to local domain.

This design uses BAR0 and BAR1 to create a 64-bit address to access the DMA Descriptor Controller. These BARs cannot connect to any other interface. If BAR0 must access a different interface, you must use an external DMA descriptor controller. Intel recommends that you select the internal DMA Descriptor Controller if you do not plan to modify this component.

The high-performance BAR2 or BAR2 and BAR3 for 64-bit addresses is available to receive data for other high performance functions.

Related Information

Getting Started with the Avalon-MM DMA on page 15

10.2. Understanding the External DMA Descriptor Controller

Using the External DMA Descriptor Controller provides more flexibility. You can either modify or replace it to meet your system requirements.You may need to modify the DMA Descriptor Controller for the following reasons:

- To implement multi-channel operation
- To implement the descriptors as a linked list or to implement a custom DMA programming model
- To store descriptors in a local memory, instead of system (host-side) memory

To interface to the DMA logic included in this variant, the custom DMA descriptor controller must implement the following functions:

- It must communicate with the Write Mover and Read Mover to copy the descriptor table to local memory.
- The Write Mover and Read Mover must execute the descriptors stored in local memory.
- The DMA Avalon-MM write (WrDCM_Master) and read (RdDCM_Master) masters must be able to update status to the TX slave (TXS).

Figure 54. Avalon-MM DMA Block Diagram with External DMA Descriptor Controller

This Platform Designer design example, `ep_g3x8_avmm256.qsys`, is available in the `<install_dir>/ ip/altera/altera_pcie/altera_pcie_a10_ed/example_design/a10` directory. This screen capture filters out some interface types for clarity.

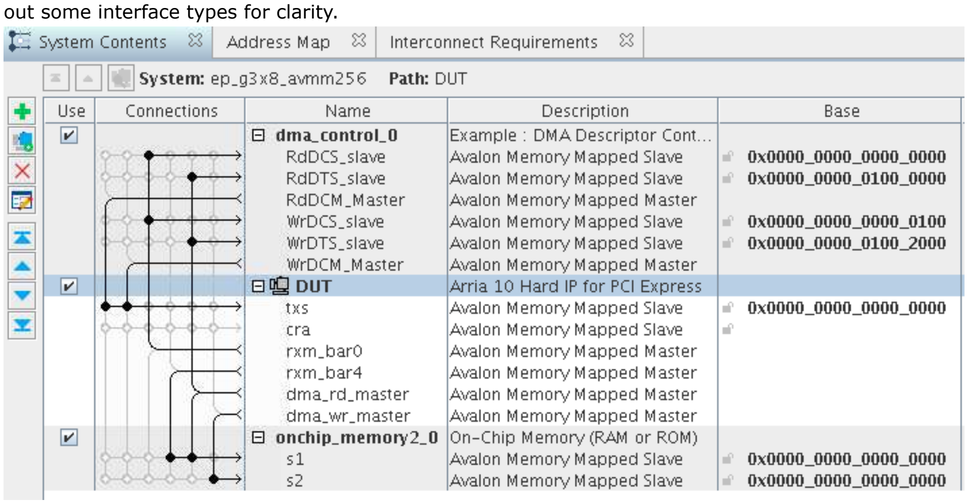
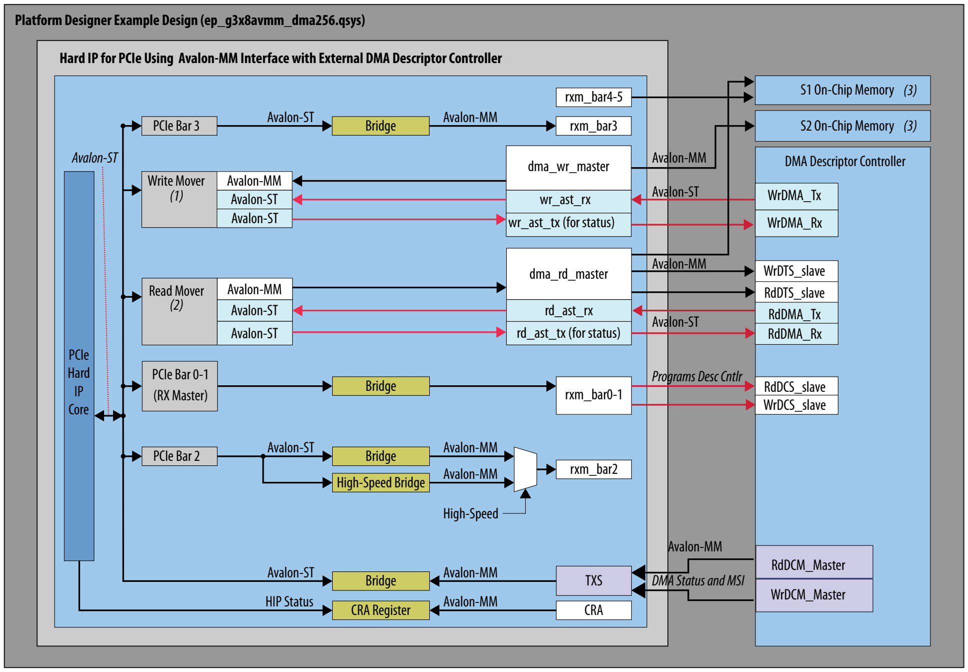


Figure 55. Avalon-MM DMA Block Diagram with External DMA Descriptor Controller

This block diagram corresponds to the Platform Designer system shown in the previous figure. When the DMA Descriptor Controller is instantiated as a external component, it drives table entries on the `RdDmaRxData_i[159:0]` and `WrDmaRxData_i[159:0]` buses.



Notes:

(1) Write Mover and dma_wr_master transfers data from local domain to the host domain.
(2) Read Mover and dma_rd_master transfer data from the host domain to local domain.
(3) This example uses on-chip memory . However, on-chip memory is not required.

The DMA modules shown in the block diagram implements the following functionality:

- Read DMA (Read Mover and dma_rd_master) –Transfers data from the host domain to the local domain. It sends Memory Read TLPs upstream and writes the completion data to external Avalon-MM components using its own high performance master port. It follows the *PCI Express Base Specification* rules concerning tags, flow control credits, read completion boundary, maximum read size, and 4 KB boundaries.
- Write DMA (Write Mover and dma_wr_master) – Transfers data from the local domain to the host domain. It reads data from an Avalon-MM slave component using its own high performance master port. It sends data upstream using Memory Write TLPs. It follows the *PCI Express Base Specification* rules concerning tags, flow control credits, RX buffer completion rules, maximum payload size, and 4 KB boundaries.
- DMA Descriptor Controller–Manages read and write DMA operations. Host software programs its internal registers with the location of the descriptor table residing in the PCI Express system memory. The descriptor control logic directs the DMA read logic to copy the entire table to the local FIFO. It then fetches the table entries from the FIFO one at a time. It directs the appropriate DMA to transfer the data between the local and host domains. It also sends DMA status upstream via the TX slave single dword port (TXS). For more information about the DMA Descriptor Controller registers, refer to DMA Descriptor Controller Registers on page 80.
- RX Master (PCIe BAR0-1) –Allows the host to program internal registers of the DMA Descriptor Controller.
- TX Slave (TXS) – The DMA Descriptor Controller reports status on each read and write descriptor to this Avalon-MM slave. It also uses this port to send MSI requests.

11. Design Implementation

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

11.1. 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 80. 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.

11.2. 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.

11.3. 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.

11.4. Warnings Encountered When Using Narrow Avalon-MM Interfaces

When the Read and Write Avalon-MM interfaces used by the Descriptor Controller Slave are 256-bit, but the Read DMA Data Master's Avalon-MM interface is narrower than 256-bit (for example, this interface is 128-bit in the Gen2x4 configuration), the Intel Arria 10 Avalon-MM DMA PCIe example design creates these two warnings:

Warning: pcie_example_design.DUT.dma_rd_master/DUT.rd_dts_slave: Master DUT.dma_rd_master cannot safely write to slave DUT.rd_dts_slave, because the master data width is narrower than the slave data width. Add byteenable support to the slave to support safe writes from a narrow master.

Warning: pcie_example_design.DUT.dma_rd_master/DUT.wr_dts_slave: Master DUT.dma_rd_master cannot safely write to slave DUT.wr_dts_slave, because the master data width is narrower than the slave data width. Add byteenable support to the slave to support safe writes from a narrow master.

The absence of byte enable support in the DTS interfaces does not cause any functional issue for the 128-bit DMA core. The controller always requests an even number of 128-bit words to the host. When completion data returns, the Platform Designer fabric combines the low and high 128-bit data forming 256 bits of data before sending it to the Descriptor Table Slave (DTS). Therefore, using byte enable masking is not needed for this application. These warnings can be safely ignored.

A. 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.

A.1. TLP Packet Formats without Data Payload

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

Figure 56. Memory Read Request, 32-Bit Addressing

Memory Read Request, 32-Bit Addressing

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="3">TC</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TD</td><td>EP</td><td colspan="2">Attr</td><td>0</td><td>0</td><td colspan="10">Length</td></tr>
<tr><td>Byte 4</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td colspan="4">Last BE</td><td colspan="4">First BE</td></tr>
<tr><td>Byte 8</td><td colspan="30">Address[31:2]</td><td>0</td><td>0</td></tr>
<tr><td>Byte 12</td><td colspan="32">Reserved</td></tr>
</table>

Figure 57. Memory Read Request, Locked 32-Bit Addressing

Memory Read Request, Locked 32-Bit Addressing

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td colspan="3">TC</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TD</td><td>EP</td><td colspan="2">Attr</td><td>0</td><td>0</td><td colspan="10">Length</td></tr>
<tr><td>Byte 4</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td colspan="4">Last BE</td><td colspan="4">First BE</td></tr>
<tr><td>Byte 8</td><td colspan="30">Address[31:2]</td><td>0</td><td>0</td></tr>
<tr><td>Byte 12</td><td colspan="32">Reserved</td></tr>
</table>

Figure 58. Memory Read Request, 64-Bit Addressing

Memory Read Request, 64-Bit Addressing

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="3">TC</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TD</td><td>EP</td><td colspan="2">Attr</td><td>0</td><td>0</td><td colspan="10">Length</td></tr>
<tr><td>Byte 4</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td colspan="4">Last BE</td><td colspan="4">First BE</td></tr>
<tr><td>Byte 8</td><td colspan="32">Address[63:32]</td></tr>
<tr><td>Byte 12</td><td colspan="30">Address[31:2]</td><td>0</td><td>0</td></tr>
</table>

Figure 59. Memory Read Request, Locked 64-Bit Addressing

Memory Read Request, Locked 64-Bit Addressing

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td colspan="3">TC</td><td>0</td><td>0</td><td>0</td><td>0</td><td>T</td><td>EP</td><td colspan="2">Attr</td><td>0</td><td>0</td><td colspan="10">Length</td></tr>
<tr><td>Byte 4</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td colspan="4">Last BE</td><td colspan="4">First BE</td></tr>
<tr><td>Byte 8</td><td colspan="32">Address[63:32]</td></tr>
<tr><td>Byte 12</td><td colspan="30">Address[31:2]</td><td>0</td><td>0</td></tr>
</table>

Figure 60. Configuration Read Request Root Port (Type 1)

Configuration Read Request Root Port (Type 1)

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TD</td><td>EP</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>Byte 4</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="4">First BE</td></tr>
<tr><td>Byte 8</td><td colspan="8">Bus Number</td><td colspan="5">Device No</td><td colspan="3">Func</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="4">Ext Reg</td><td colspan="6">Register No</td><td>0</td><td>0</td></tr>
<tr><td>Byte 12</td><td colspan="32">Reserved</td></tr>
</table>

Figure 61. I/O Read Request

I/O Read Request

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TD</td><td>EP</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>Byte 4</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="4">First BE</td></tr>
<tr><td>Byte 8</td><td colspan="30">Address[31:2]</td><td>0</td><td>0</td></tr>
<tr><td>Byte 12</td><td colspan="32">Reserved</td></tr>
</table>

Figure 62. Message without Data

Message without Data

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>r
2</td><td>r
1</td><td>r
0</td><td>0</td><td colspan="3">TC</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TD</td><td>EP</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>Byte 4</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td colspan="8">Message Code</td></tr>
<tr><td>Byte 8</td><td colspan="32">Vendor defined or all zeros</td></tr>
<tr><td>Byte 12</td><td colspan="32">Vendor defined or all zeros</td></tr>
</table>

Note:

(1) Not supported in Avalon-MM.

Figure 63. Completion without Data

Completion without Data

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td colspan="3">TC</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TD</td><td>EP</td><td colspan="2">Attr</td><td>0</td><td>0</td><td colspan="10">Length</td></tr>
<tr><td>Byte 4</td><td colspan="16">Completer ID</td><td colspan="3">Status</td><td>B</td><td colspan="12">Byte Count</td></tr>
<tr><td>Byte 8</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td>0</td><td colspan="7">Lower Address</td></tr>
<tr><td>Byte 12</td><td colspan="32">Reserved</td></tr>
</table>

Figure 64. 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																															

A.2. TLP Packet Formats with Data Payload

Figure 65. 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 66. 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 67. 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 68. 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 69. Completion with Data

Completion with Data

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td colspan="3">TC</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TD</td><td>EP</td><td colspan="2">Attr</td><td>0</td><td>0</td><td colspan="10">Length</td></tr>
<tr><td>Byte 4</td><td colspan="16">Completer ID</td><td colspan="3">Status</td><td>B</td><td colspan="12">Byte Count</td></tr>
<tr><td>Byte 8</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td>0</td><td colspan="7">Lower Address</td></tr>
<tr><td>Byte 12</td><td colspan="32">Reserved</td></tr>
</table>

Figure 70. Completion Locked with Data

Completion Locked with Data

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td colspan="3">TC</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TD</td><td>EP</td><td colspan="2">Attr</td><td>0</td><td>0</td><td colspan="10">Length</td></tr>
<tr><td>Byte 4</td><td colspan="16">Completer ID</td><td colspan="3">Status</td><td>B</td><td colspan="12">Byte Count</td></tr>
<tr><td>Byte 8</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td>0</td><td colspan="7">Lower Address</td></tr>
<tr><td>Byte 12</td><td colspan="32">Reserved</td></tr>
</table>

Figure 71. Message with Data

Message with Data

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Byte 0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>r2</td><td>r1</td><td>r0</td><td>0</td><td colspan="3">TC</td><td>0</td><td>0</td><td>0</td><td>0</td><td>TD</td><td>EP</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="10">Length</td></tr>
<tr><td>Byte 4</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td colspan="8">Message Code</td></tr>
<tr><td>Byte 8</td><td colspan="32">Vendor defined or all zeros for Slot Power Limit</td></tr>
<tr><td>Byte 12</td><td colspan="32">Vendor defined or all zeros for Slots Power Limit</td></tr>
</table>

B. Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA 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	Intel Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide
16.1.1	Intel Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide
16.1	Intel Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide
16.0	Intel Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide
15.1	Intel Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide
15.0	Intel Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide
14.1	Intel Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide

C. Document Revision History

C.1. Document Revision History for the Intel Arria 10 or Intel Cyclone 10 GX Avalon-MM DMA Interface for PCIe Solutions User Guide

Date	Version	Changes Made
2019.05.23	18.0.1	Added a 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.
2019.04.30	18.0.1	Updated Table 3 to show that the Avalon-MM DMA feature is not supported in Root Port mode.
2018.08.28	18.0.1	Added the step to invoke vsim to the instructions for simulating the example design in ModelSim.
2018.06.15	18.0.1	Added note that Flush reads are not supported when burst mode for BAR2 is enabled. Updated the list of configurations supported by the Avalon-MM and Avalon-MM with DMA variants.
2018.05.07	18.0	Changed all references to Intel Cyclone 10 to Intel Cyclone 10 GX.
2017.10.06	17.1	Made the following change to the user guide: • Added support for Intel Cyclone 10 GX devices. • Added optional parameter to invert the RX polarity. • 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 completions. • Added missing sequence of programming steps in *DMA Descriptor Controller Registers*. • Rebranded as Intel. • Corrected minor errors and typos.
2017.05.26	17.0	Made the following changes to the user guide: • Added note that starting with the Intel Quartus Prime Pro Edition 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.
2017.05.08	17.0	Made the following changes to 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:

continued...

Date	Version	Changes Made
		• Updated *PCI Express Gen3 Bank Usage Restrictions* status. These restrictions affect all Aria 10 ES and production devices. • Corrected *Table 2: Comparison for 128- and 256-Bit Avalon-MM with DMA Interface to the Application Layer*. The 128-bit interface supports Gen3 x2 operation. • Clarified behavior of the Read Descriptor Controller and Write Descriptor Controller Avalon-MM Master interfaces. These Controllers send an MSI to the host upon completion of the last descriptor unless MSIs are disabled. By default, MSIs are enabled. • Corrected *Comparison of Avalon-ST, Avalon-MM and Avalon-MM with DMA Interfaces* table. Out-of-order Completions are not supported transparently for the Avalon-MM with DMA interface. • Corrected default values for the *Uncorrectable Internal Error Mask Register* and *Correctable Internal Error Mask Register* registers. • Added *Understanding the Avalon-MM DMA Ports* to the *Getting Started with the Avalon-MM DMA Endpoint* chapter. • Corrected minor errors and typos.
2017.03.15	16.1.1	Made the following changes: • 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.
2016.10.28	16.1	Made the following to the IP core changes: • Increased the max DMA transfer to 1 MB for both the 128- and 256-bit interfaces. • Timing models are now final for most Intel Arria 10 device packages. Exceptions include some military and automotive speed grades with extended temperature ranges. Made the following changes to the user guide: • Changed the recommended value of `test_in[31:0]` from 0xa8 to 0x188. • Removed recommendations about connecting `pin_perst`. These recommendations do not apply to Arria 10 devices. • Corrected the number of tags supported in the *Feature Comparison for all Hard IP for PCI Express IP Cores* table. • Added PCIe bifurcation to the *Feature Comparison for all Hard IP for PCI Express IP Cores* table. PCI bifurcation is not supported. • Removed reference to a Linux software driver for the DMA modules which is not available. • Added section covering design example limitations. • Added -3 to recommended speed grades for the 125 MHz interface.
2016.05.02	16.0	Redesigned the 128-bit interface to the Application Layer resulting in consistently high throughput, for both on-chip and external memory. In the *Getting Started with the Avalon-MM DMA Endpoint* 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 support for Intel FPGA IP Evaluation Mode in the Quartus Prime Pro Edition software. Added simulation support for Gen3 PIPE mode using the ModelSim, VCS, and NCSim simulators. Added automatic generation of basic Signal Tap Logic Analyzer files to facilitate debugging. Revised discussion of the DMA Descriptor Controller in the *Avalon-MM with DMA IP Core Architecture*. Revised *Read DMA Example* to reflect current maximum transfer size of 64 KB for 256-bit interface. The example now corresponds to an example design provided in the <*install_dir*> Updated figures in *Physical Layout of Hard IP in Intel Arria 10 Devices* to include more detail about transceiver banks and channel restrictions.

continued...

Date	Version	Changes Made
		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. Removed *Intel Arria 10 PCI Express Quick Start Guide* chapter. This chapter does not provide DMA functionality. Corrected description of Write Descriptor Table Avalon-MM Slave Port. Added *Vendor Specific Extended Capability (VSEC)* parameter descriptions which were missing from previous versions. Added transceiver bank usage placement restrictions for Gen3 ES3 devices. Removed support for -3 speed grade devices. Added appendix listing previous versions of this user guide. Corrected minor errors and typos.
2015.11.02	15.1	Made the following changes: • Added support for 256 tags to enhance throughput in high latency designs. • Added support for RX Completion buffer overflow monitoring. • To enhance performance and reduce internal buffering requirements, limited descriptor size to 8 KB. • Redesigned component GUI. • Added new Design Example tab that you can use to generate a design example you can download to the Altera Intel Arria 10 GX FPGA Development Kit. • Removed the parameter values **High** and **Maximum** from the RX buffer allocation parameter. These values are not supported for the Avalon-MM interface. • Enhanced the definition of `npor`. • Corrected resource utilization. • Clarified that conditions necessary before changing the base address for `RC Read Status and Descriptor Base (Low)` and `RC Write Status and Descriptor Base (Low)` registers. • Added an immediate write mode for single dword writes. The data is stored in the `WR_RC_LOW_SRC_ADDR` register. The new `Immediate Write Mode` bit of the DMA Descriptor controls this functionality. • Corrected *TLP Support Comparison for all Hard IP for PCI Express IP Cores* entries. Only Completions with and without data are supported for the Avalon-MM DMA interface. Message Requests with and without data are not supported for the Avalon-M interface. • Added optional Hard IP Status bus signals to the *Avalon-MM DMA Bridge with Internal Descriptor Controller* and *Avalon-MM DMA Bridge with External Descriptor Controller* figures.
2015.06.05	15.0	Added note in*Physical Layout of Hard IP in Intel 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	• Added **Enable Altera Debug Master Endpoint (ADME)** parameter to support optional Native PHY register programming with the Altera System Console. • Added support for downstream burst read request for a payload of size up to 4 KB, if **Enable burst capability for RXM BAR2 port** is turned on in the Parameter Editor. Previous maximum downstream read request payload size was 512 bytes. • Corrected the allowed value of the **Maximum payload size** parameter for Avalon-MM DMA IP core variations, in *Device Capabilities* topic. • Corrected the supported variations to include Gen3 x2.

continued...

Date	Version	Changes Made
		• Removed the **High** and **Maximum** values for the **RX Buffer credit allocation -performance for received requests** parameter. These values are no longer valid settings. />. • 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 Intel Arria 10 Devices*. . • Reinstated *Design Implementation* chapter. • 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. section. • Removed Migration and TLP Format appendices, and added new *Frequently Asked Questions*appendix. • Updated information in *SDC Timing Constraints* section. • Removed list of static example designs from *Design Examples*. You can derive the list from the installation directory where example designs are available. • Fixed minor errors and typos.
2014.12.15	14.1	Made the following changes to the Intel Arria 10 user guide: • In the *Getting Started* chapter, corrected directory path for the simulation. • Added the fact that the RX Burst Master only support dword granularity. • Added definitions for `test_in[2]`, `test_in[6]` and `test_in[7]`. • Added instructions for Quartus II compilation.
2014.08.18	13.1 Intel Arria 10	Made the following changes to the Intel Arria 10 Avalon-MM DMA for PCI Express IP core: • Revised programming model for the Descriptor Controller. • 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 that generates `altpcie_monitor_a10_dlhip_tlp_file_log.log`. Refer to *Understanding Simulation Dump File Generation* for details. • Added support for either 128- or 256-bit interface to the Application Layer. • Added support for 64-bit addressing, making address translation unnecessary. • Removed *Channel Placement for PCIe in Intel Arria 10 Devices*. Please contact your Altera sales representative for PLL and channel usage. • Added support for optional bursting RX Master for BAR2. • Revised *Read DMA Example* and *Software Program for Simultaneous Read and Write DMA* to work with revised programming model for the Descriptor Controller. • Added the following optimizations for the Descriptor Controller: — Optimized performance for smaller payloads such as 64-byte Ethernet packets — Reduced overhead for host updates — Support for concurrent dynamic host updates and DMA operation — Support for choice to embed Descriptor Controller in the Avalon-MM bridge or instantiate separately • Added access to selected Configuration Space registers and link status registers through the optional Control Register Access (CRA) Avalon-MM slave port. • Added simulation support for Phase 2 and Phase 3 equalization when requested by third-party BFM for Gen3 variants. • Added multiple MSI/MSI-X support.

continued...

Date	Version	Changes Made
		Made the following changes to the user guide: • Removed 125 MHz clock as optional `refclk` frequency in Intel Arria 10 devices. Intel Arria 10 devices support a 100 MHz reference clock as specified by the *PCI Express Base Specification, Rev 3.0* • Corrected values for **Maximum payload size** parameter. The sizes available are 128 or 256 bytes. • Enhanced definition of Device ID and Sub-system Vendor ID to say that these registers are only valid in the Type 0 (Endpoint) Configuration Space. • 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 *Next Steps in Creating a Design for PCI Express* to *Datasheet* chapter. • Removed the *Transaction Layer Protocol Details* chapter. This information only applies to the Avalon-ST interface. • Removed `txdatavalid0` signal from the PIPE interface. This signal is not available. • Removed references to the MegaWizard® Plug-In Manager. In 14.0 the IP Parameter Editor Powered by Platform Designer has replaced the MegaWizard Plug-In Manager. • Added definitions for `test_in[2]`, `test_in[6]` and `test_in[7]`. • Corrected interface widths in the *Performance and Resource Utilization Intel Arria 10 Avalon-MM DMA for PCI Express* table in the *Datasheet: Intel Arria 10 Avalon-MM DMA for PCIe* chapter. • Removed discussion of `pclk`. This clock is not customer accessible in Intel Arria 10 devices. • Corrected *Reset Controller in Intel Arria 10 Devices* figure in *Reset and Clocks* chapter. • Corrected bit definitions for `CvP Status` register. • Removed PLL from channel placement figures. • Added fast passive parallel (FPP) to supported configuration schemes in *CvP in Intel Arria 10 Devices* figure. • Updated *Power Supply Voltage Requirements* table. • Corrected the name of the Descriptor Instructions bus. The letters *DMA* are now *Ast*. For example `WrDMARXValid_i` is now `WrAstRXValid_i`. • Added `RD_CONTROL` and `WR_CONTROL` register `Done` bit. When set, the Descriptor Controller writes this bit for each descriptor in the status table and sends a single MSI interrupt after the final descriptor completes. • Removed the following chapters that have minimal relevance to the Intel Arria 10 Avalon-MM DMA Interface IP Core. These chapters are available in the more comprehensive Avalon-ST versions : — *Design Implementation* — *Optional Features* — *Debugging* — *Throughput Optimization*
2013.12.02	13.1 Intel Arria 10	Initial release.