Intel® L- and H-tile Avalon® Memory-mapped+ IP for PCI Express* User Guide

Updated for Intel® Quartus® Prime Design Suite: **23.4**

Online Version
Send Feedback

UG-20170

683527
2024.03.05

Contents

1. Introduction

This User Guide is applicable to H-Tile and L-Tile variants of the Stratix® 10 devices.

Related Information

- Introduction to Intel FPGA IP Cores
  Provides general information about all Intel FPGA IP cores, including parameterizing, generating, upgrading, and simulating IP cores.
- Creating a Combined Simulator Setup Script
  Create simulation scripts that do not require manual updates for software or IP version upgrades.
- Managing Quartus Prime Projects
  Guidelines for efficient management and portability of your project and IP files.
- Stratix 10 Avalon-ST Hard IP for PCIe Solutions User Guide
  Use this link for signal definitions that are shared with the Stratix 10 Avalon-ST Hard IP for PCIe IP core.

1.1. Features

The Intel L-/H-Tile Avalon-MM+ for PCI Express IP supports the following features:

- Complete protocol stack including the Transaction, Data Link, and Physical Layers implemented as a hard IP.
- Support for Gen3 x16 for Endpoints.
- Support for 512-bit Avalon-MM interface to the Application Layer at the Gen3 x16 data rate for Stratix 10 devices.
- Support for address widths ranging from 10-bit to 64-bit for the Avalon-MM interface to the Application Layer.
- Platform Designer design example demonstrating parameterization, design modules, and connectivity.
- Standard Avalon®-MM interfaces:
  - High-throughput bursting Avalon-MM slave with byte enable support.
  - High-throughput bursting Avalon-MM master with byte enable support associated with 1 - 7 Base Address Registers (BARs).
- High-throughput data movers for DMA support:
  - Moves data from local memory in Avalon-MM space to system memory in PCIe* space using PCIe Memory Write (MWr) Transaction Layer Packets (TLPs).
  - Moves data from system memory in PCIe space to local memory in Avalon-MM space using PCIe Memory Read (MRd) TLPs.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- The Intel L-/H-Tile Avalon-MM+ for PCI Express IP supports the Separate Reference Clock With No Spread Spectrum architecture (SRNS), but not the Separate Reference Clock With Independent Spread Spectrum architecture (SRIS)
- Support for legacy interrupts (INTx), Message Signaled Interrupts (MSI) and MSI-X.
- Advanced Error Reporting (AER): In Stratix 10 devices, Advanced Error Reporting is always enabled in the PCIe Hard IP for both the L and H transceiver tiles.
- Completion timeout checking.
- Modular implementation to select the required features for a specific application:
  - Simultaneous support for data movers and high-throughput Avalon-MM slaves and masters.
  - Avalon-MM slave to easily access the entire PCIe address space without requiring any PCIe specific knowledge.
- Autonomous Hard IP mode, allowing the PCIe IP core to begin operation before the FPGA fabric is programmed. This mode is enabled by default. It cannot be disabled.

  Note: Unless Readiness Notifications mechanisms are used (see Section 6.23 of the *PCIe Base Specification*), the Root Complex and/or system software must allow at least 1.0 s after a Conventional Reset of a device before it may determine that a device which fails to return a Successful Completion status for a valid Configuration Request is a broken device. This period is independent of how quickly Link training completes.
- Available in Quartus® Prime Pro Edition, in both Platform Designer and IP Catalog.
- Operates at 250 MHz in -1 or -2 speed grade Stratix 10 devices.
- Easy to use:
  - No license requirement.

Note: For a list of differences between this Intel L-/H-Tile Avalon-MM+ for PCI Express IP and the Intel L-/H-Tile Avalon-MM for PCI Express IP (which can support configurations up to Gen3 x8), refer to the *Avalon-MM IP Variants Comparison* section in the Appendix.

Note: Throughout this document, the term Avalon-MM Hard IP+ for PCI Express may also be used to refer to the Intel L-/H-Tile Avalon-MM+ for PCI Express IP.

1.2. Device Family Support

The following terms define IP core support levels for Intel® FPGA IP cores in Stratix 10 devices:

- **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 1. Device Family Support

Device Family	Support Level
Stratix 10	Final
Other device families	No support

Note: For IP cores that support multiple device families, list the device families sequentially, in alphabetical order.

Related Information

Timing and Power Models
Reports the default device support levels in the current version of the Quartus Prime Pro Edition software.

1.3. Release Information

Table 2. Release information for the Intel L-/H-Tile Avalon-MM+ for PCI Express IP

Item	Description
Version	Quartus Prime Pro Edition 18.0 Software Release
Release Date	May 2018

1.4. Recommended Speed Grades

Recommended speed grades are pending characterization of production Stratix 10 devices.

Table 3. Recommended Speed Grades

Lane Rate	Link Width	Interface Width (bits)	Application Clock Frequency (MHz)	Recommended Speed Grades
Gen3 (8Gbps)	x16	512	250	-1, -2

Link to the appropriate *Timing Closure and Optimization* topic in the Quartus Prime Standard Edition or Quartus Prime Pro Edition Handbook. Include this link to *Setting up and Running Analysis and Synthesis* in *Quartus Prime Help*.

Related Information

Quartus Standard to Timing Closure and Optimization
Use this link for the Quartus Prime Pro Edition Software.

1.5. Resource Utilization

For soft IP—Consult with Marketing for a list of configurations that they consider the most important. Round the numbers for ALMs and logic registers up to the nearest 50.

For hard IP—list resource utilization for any bridge or wrapper that is necessary to interface to the IP. Include the following statement:

The following table shows the typical device resource utilization for selected configurations using version *18.0* of the Quartus Prime Pro Edition software. The number of ALMs and logic registers are rounded up to the nearest 50. The number stated for M20K memory blocks includes no rounding.

Note: The resource utilization numbers in the following table are for the three variations of the available Gen3 x16 design example: DMA, BAS, and PIO. For more details on the design example and these variations, refer to the chapter *Design Example and Test Bench*.

Table 4. Resource Utilization

Variation	ALMs	M20K Memory Blocks	Logic Registers
DMA	57,099	571	119,054
BAS	45,236	485	91,154
PIO	36,754	384	72,365

Related Information

Design Compilation
To understand how the Quartus Prime software compiles and synthesizes your RTL design.

1.6. Transceiver Tiles

Stratix 10 introduces several transceiver tile variants to support a wide variety of protocols.

Figure 1. Stratix 10 Transceiver Tile Block Diagram

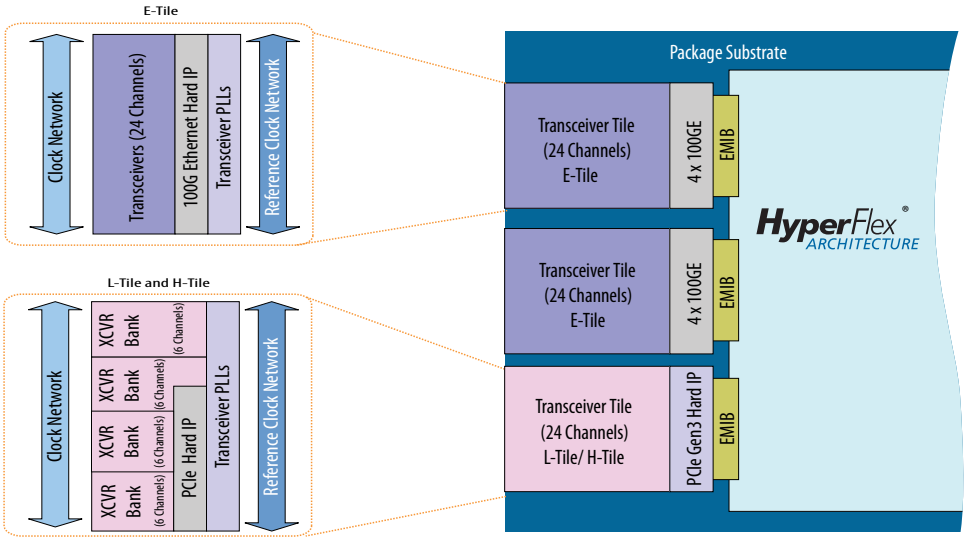


Table 5. Transceiver Tiles Channel Types

Tile	Device Type	Channel Capability		Channel Hard IP Access
		Chip-to-Chip	Backplane	
L-Tile	GX/SX	26 Gbps (NRZ)	12.5 Gbps (NRZ)	PCIe Gen3x16
H-Tile	GX/SX/MX/TX	28.3 Gbps (NRZ)	28.3 Gbps (NRZ)	PCIe Gen3x16

Note: E-Tiles do not include any PCIe Hard IP block and do not support PCIe, so they are not discussed in this document.

L-Tile and H-Tile

Both L and H transceiver tiles contain four transceiver banks-with a total of 24 duplex channels, eight ATX PLLs, eight fPLLs, eight CMU PLLs, a PCIe Hard IP block, and associated input reference and transmitter clock networks. L and H transceiver tiles also include 10GBASE-KR/40GBASE-KR4 FEC block in each channel.

L-Tiles have transceiver channels that support up to 26 Gbps chip-to-chip or 12.5 Gbps backplane applications. H-Tiles have transceiver channels to support 28.3 Gbps applications. H-Tile channels support fast lock-time for Gigabit-capable passive optical network (GPON).

Only Stratix 10 GX/SX devices support L-Tiles, while most of the product line supports H-Tiles. Package migration is available with Stratix 10 GX/SX from L-Tile to H-Tile variants.

1.7. Channel Availability

PCIe Hard IP Channel Restrictions

Each L- or H-Tile transceiver tile contains one PCIe Hard IP block. The following table and figure show the possible PCIe Hard IP channel configurations, the number of unusable channels, and the number of channels available for other protocols.

Figure 2. PCIe Hard IP Channel Configurations Per Transceiver Tile

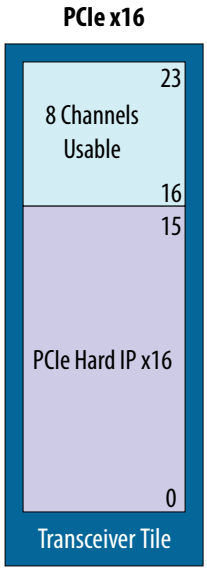


Table 6. Unusable Channels

PCIe Hard IP Configuration	Number of Unusable Channels Remaining in the Tile	Usable Channels Remaining in the Tile
PCIe x8	0	16
PCIe x16	0	8

The table below maps all transceiver channels to the PCIe Hard IP channels in the available tiles.

Table 7. PCIe Hard IP channel mapping across all tiles

Tile Channel Sequence	PCIe Hard IP Channel	Index within I/O Bank	Bottom Left Tile Bank Number	Top Left Tile Bank Number	Bottom Right Tile Bank Number	Top Right Tile Bank Number
23	N/A	5	1F	1N	4F	4N
22	N/A	4	1F	1N	4F	4N
21	N/A	3	1F	1N	4F	4N

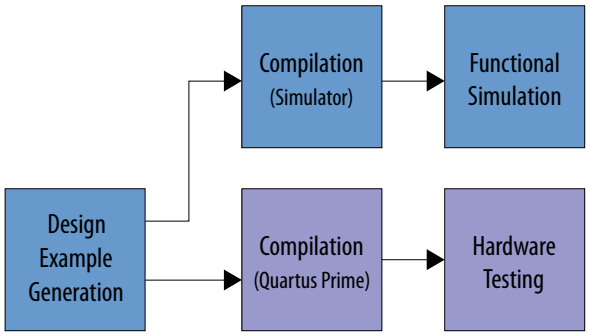
continued...

Tile Channel Sequence	PCIe Hard IP Channel	Index within I/O Bank	Bottom Left Tile Bank Number	Top Left Tile Bank Number	Bottom Right Tile Bank Number	Top Right Tile Bank Number
20	N/A	2	1F	1N	4F	4N
19	N/A	1	1F	1N	4F	4N
18	N/A	0	1F	1N	4F	4N
17	N/A	5	1E	1M	4E	4M
16	N/A	4	1E	1M	4E	4M
15	15	3	1E	1M	4E	4M
14	14	2	1E	1M	4E	4M
13	13	1	1E	1M	4E	4M
12	12	0	1E	1M	4E	4M
11	11	5	1D	1L	4D	4L
10	10	4	1D	1L	4D	4L
9	9	3	1D	1L	4D	4L
8	8	2	1D	1L	4D	4L
7	7	1	1D	1L	4D	4L
6	6	0	1D	1L	4D	4L
5	5	5	1C	1K	4C	4K
4	4	4	1C	1K	4C	4K
3	3	3	1C	1K	4C	4K
2	2	2	1C	1K	4C	4K
1	1	1	1C	1K	4C	4K
0	0	0	1C	1K	4C	4K

2. Quick Start Guide

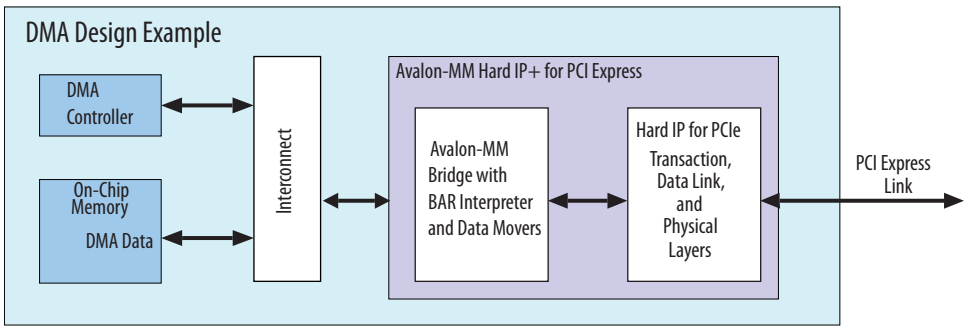
Using Quartus Prime Pro Edition, you can generate a simple DMA design example for the Intel L-/H-Tile Avalon-MM+ for PCI Express IP. The generated design example reflects the parameters that you specify. It automatically creates the files necessary to simulate and compile in the Quartus Prime Pro Edition software. You can download the compiled design to the Stratix 10 Development Board. To download to custom hardware, update the Quartus Prime Pro Edition Settings File (`.qsf`) with the correct pin assignments .

Figure 3. Development Steps for the Design Example



2.1. Design Components

Figure 4. Block Diagram for the DMA PCIe Design Example



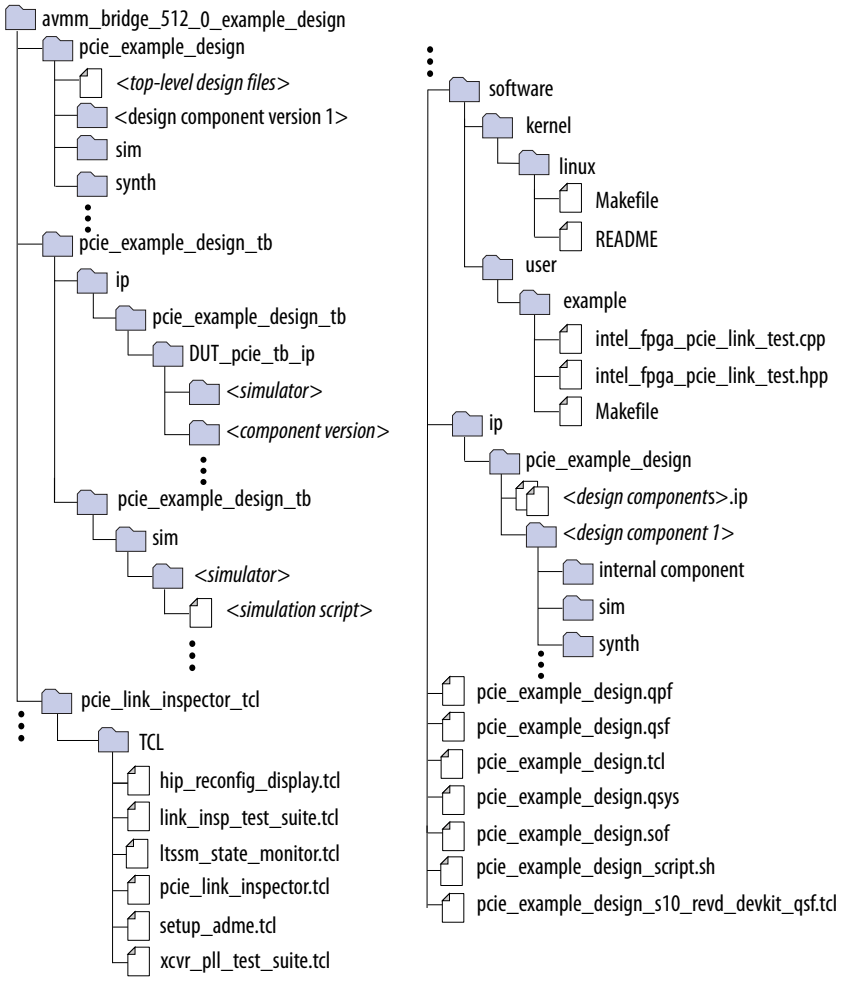
Note: The DMA design example includes an external DMA controller, which is instantiated outside of the Intel L-/H-Tile Avalon-MM+ for PCI Express IP core. If you are not using the DMA design example, you need to implement your own external DMA controller.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

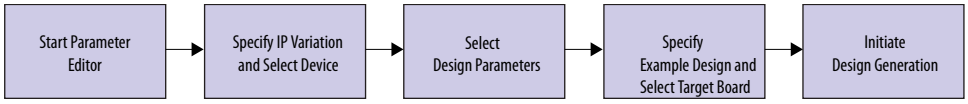
2.2. Directory Structure

Figure 5. Directory Structure for the Generated Design Example



2.3. Generating the Design Example

Figure 6. Procedure

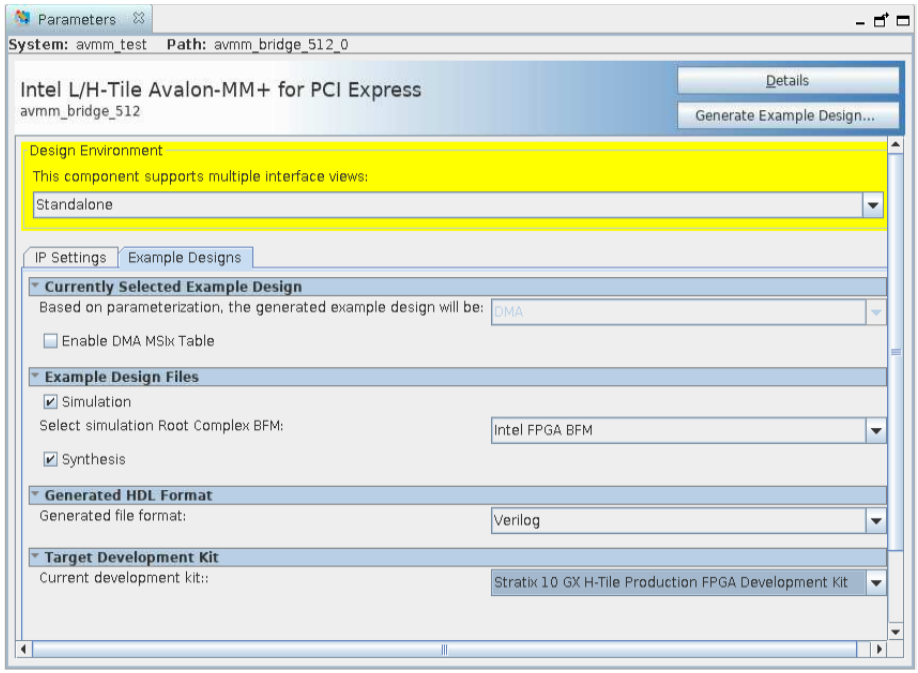


1. In the Quartus Prime Pro Edition software, create a new project (**File ➤ New Project Wizard**).
2. Specify the **Directory, Name**, and **Top-Level Entity.**
3. For **Project Type**, accept the default value, **Empty project**. Click **Next**.
4. For **Add Files** click **Next**.

5. For **Family, Device & Board Settings** under **Family**, select **Stratix 10 (GX/SX/MX/TX)** and the **Target Device** for your design. Note that the selected device is only used if you select **None** in Step 10e below.
6. Click **Finish**.
7. In the IP Catalog locate and add the **Intel L-/H-Tile Avalon-MM+ for PCI Express** IP.
8. In the **New IP Variant** dialog box, specify a name for your IP. Click **Create**.
9. On the **IP Settings** tabs, specify the parameters for your IP variation.
10. On the **Example Designs** tab, make the following selections:
    a. For **Available Example Designs,** select **DMA**.
    b. For **Example Design Files**, turn on the **Simulation** and **Synthesis** options. If you do not need these simulation or synthesis files, leaving the corresponding option(s) turned off significantly reduces the example design generation time.
    c. For **Select simulation Root Complex BFM**, choose the appropriate BFM:
        - **Intel FPGA BFM**: This bus functional model (BFM) supports x16 configurations by downtraining to x8.
        - **Third-party BFM**: If you want to simulate all 16 lanes, use a third-party BFM. Refer to AN-811: Using the Avery BFM for PCI Express Gen3x16 Simulation on Stratix 10 Devices for information about simulating with the Avery BFM.
    d. For **Generated HDL Format**, only Verilog is available in the current release.
    e. For **Target Development Kit**, select the appropriate option.

    Note: If you select **None**, the generated design example targets the device specified. Otherwise, the design example uses the device on the selected development board. If you intend to test the design in hardware, make the appropriate pin assignments in the `.qsf` file.
11. Select **Generate Example Design** to create a design example that you can simulate and download to hardware. If you select one of the Stratix 10 development boards, the device on that board supersedes the device previously selected in the Quartus Prime Pro Edition project if the devices are different. When the prompt asks you to specify the directory for your example design, you can choose to accept the default directory, *<example_design>*`/avmm_bridge_512_0_example_design`

Figure 7. Example Design Tab



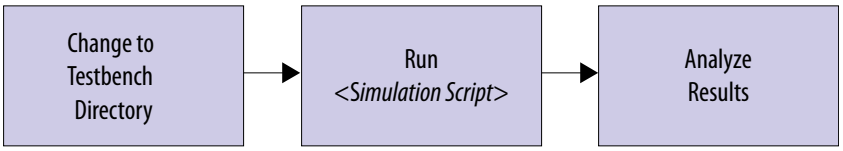
When you generate an Stratix 10 example design, a file called `recommended_pinassignments_s10.txt` is created in the directory avmm_bridge_512_0_example_design. (1)

12. Click **Finish** to close the dummy .ip file, which does not have the complete configuration.
13. Click **No** upon receiving the prompt, **Recent changes have not been generated. Generate now?** since you do not need to generate any file for the design associated with the dummy .ip file.
14. Change directory to the example design directory. Open pcie_example_design.qpf in Quartus Prime Pro Edition.
15. Start compilation. This generates the .sof file for the complete example design, which you can download to a board to perform hardware verification.
16. Close your project.

(1) This file contains the recommended pin assignments for all the pins in the example design. If you chose a GX, SX or MX development kit option in the pull-down menu for **Target Development Kit**, the pin assignments in the `recommended_pinassignments_s10.txt` file match those that are in the `.qsf` file in the same directory. If you chose **NONE** in the pull-down menu, the `.qsf` file does not contain any pin assignment. In this case, you can copy the pin assignments in the `recommended_pinassignments_s10.txt` file to the `.qsf` file. You can always change any pin assignment in the `.qsf` file to satisfy your design or board requirements.

2.4. Simulating the Design Example

Figure 8. Procedure



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

Table 8. Steps to Run Simulation

Simulator	Working Directory	Instructions
Questa Intel FPGA Edition	*<example_design>*/pcie_example_design_tb/pcie_example_design_tb/sim/mentor/	1. Invoke vsim (by typing `vsim`, which brings up a console window where you can run the following commands). 2. `do msim_setup.tcl` *Note:* Alternatively, instead of doing Steps 1 and 2, you can type: `vsim -c -do msim_setup.tcl`. 3. `ld_debug` 4. `run -all` 5. A successful simulation ends with the following message, "Simulation stopped due to successful completion!"
ModelSim*	*<example_design>*/pcie_example_design_tb/pcie_example_design_tb/sim/mentor/	1. Invoke vsim (by typing `vsim`, which brings up a console window where you can run the following commands). 2. `do msim_setup.tcl` *Note:* Alternatively, instead of doing Steps 1 and 2, you can type: `vsim -c -do msim_setup.tcl`. 3. `ld_debug` 4. `run -all` 5. A successful simulation ends with the following message, "Simulation stopped due to successful completion!"

continued...

Simulator	Working Directory	Instructions
VCS*	<example_design>/pcie_example_design_tb/pcie_example_design_tb/sim/synopsys/vcs	1. sh vcs_setup.sh USER_DEFINED_COMPILE_OPTIONS="" USER_DEFINED_ELAB_OPTIONS="-xlrm\ uniq_prior_final" USER_DEFINED_SIM_OPTIONS="" 2. A successful simulation ends with the following message, "Simulation stopped due to successful completion!"
NCSim*	<example_design>/pcie_example_design_tb/pcie_example_design_tb/sim/cadence	1. sh ncsim_setup.sh USER_DEFINED_SIM_OPTIONS="" USER_DEFINED_ELAB_OPTIONS ="-timescale\ 1ns/1ps" 2. A successful simulation ends with the following message, "Simulation stopped due to successful completion!"
Xcelium* Parallel Simulator	<example_design>/pcie_example_design_tb/pcie_example_design_tb/sim/xcelium	1. sh xcelium_setup.sh USER_DEFINED_SIM_OPTIONS="" USER_DEFINED_ELAB_OPTIONS ="-timescale\ 1ns/1ps\ -NOWARN\ CSINFI" 2. A successful simulation ends with the following message, "Simulation stopped due to successful completion!"

The DMA testbench for the design example completes the following tasks:

1. Instructs the Read Data Mover to fetch the descriptors for the DMA Read operation from the PCI Express* system memory.
2. The Read Data Mover reads the data from the PCI Express system memory, and writes the data to the memory in Avalon-MM address space according to the descriptors fetched in Step 1.
3. Instructs the Read Data Mover to fetch the descriptors for the DMA Write operation from the PCI Express system memory.
4. The Write Data Mover reads the data from the memory in the Avalon-MM address space, and writes the data to the PCI Express system memory according to the descriptors fetched in Step 3.
5. Compares the source data read from system memory in Step 2 to the data written back to the system memory in Step 4.

The simulation reports, "Simulation stopped due to successful completion" if no errors occur.

Figure 9. Partial Transcript from Successful Simulation Testbench

```
INFO:          106776 ns        Maximum Link Width: x8
INFO:          106776 ns      Supported Link Speed: 8.0GT/s or 5.0GT/s or 2.5GT/s
INFO:          106776 ns                 L0s Entry: Supported
INFO:          106776 ns                 L1  Entry: Not Supported
INFO:          106776 ns          L0s Exit Latency: 2 us to 4 us
INFO:          106776 ns          L1  Exit Latency: Less Than 1 us
INFO:          118336 ns BAR Address Assignments:
INFO:          118336 ns BAR    Size       Assigned Address  Type
INFO:          118336 ns ---    ----       ----------------
INFO:          118336 ns BAR1:0  64 KBytes 00000000 80000000 Prefetchable
INFO:          118336 ns BAR2   Disabled
INFO:          118336 ns BAR3   Disabled
INFO:          118336 ns BAR4   Disabled
INFO:          118336 ns BAR5   Disabled
INFO:          118336 ns ExpROM Disabled
INFO:          119136 ns
INFO:          119136 ns Completed configuration of Endpoint BARs.
INFO:          120064 ns Initializing RP Memory for DMA-RD ....
INFO:          120064 ns RP Memory Initialization Done!
INFO:          120064 ns Starting DMA Read....
INFO:          130072 ns Starting DMA Write....
INFO:          131080 ns Passed: 0128 same bytes in BFM mem addr 0x01002000 and 0x01000000
Finished comparison!
SUCCESS: Simulation stopped due to successful completion!
Simulation passed
```

Related Information

AN-811: Using the Avery BFM for PCI Express Gen3x16 Simulation on Stratix 10 Devices

This document describes how to use the Avery BFM to run PCI Express Gen3x16 simulations on Stratix 10 devices.

2.5. Compiling the Design Example and Programming the Device

1. Navigate to `<project_dir>/avmm_bridge_512_0_example_design/` and open `pcie_example_design.qpf`.
2. On the Processing menu, select **Start Compilation**.
3. After successfully compiling your design, program the targeted device with the Programmer.

2.6. Installing the Linux Kernel Driver

Before you can test the design example in hardware, you must install the Linux kernel driver. You can use this driver to perform the following tests:

- A PCIe link test that performs 100 writes and reads
- Memory space DWORD[2] reads and writes
- Configuration Space DWORD reads and writes

[2] 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.

In addition, you can use the driver to change the value of the following parameters:

- The BAR being used
- The selected device by specifying the bus, device and function (BDF) numbers for the required device

Complete the following steps to install the kernel driver:

1. Navigate to `./software/kernel/Linux` under the example design generation directory.
2. Change the permissions on the `install`, `load`, and `unload` files:

   ```
   $ chmod 777 install load unload
   ```
3. Install the driver:

   ```
   $ sudo ./install
   ```
4. Verify the driver installation:

   ```
   $ lsmod | grep intel_fpga_pcie_drv
   ```

   Expected result:

   ```
   intel_fpga_pcie_drv 17792 0
   ```
5. Verify that Linux recognizes the PCIe design example:

   ```
   $ lspci -d 1172:000 -v | grep intel_fpga_pcie_drv
   ```

   Note: If you have changed the Vendor ID, substitute the new Vendor ID for Intel's Vendor ID in this command.

   Expected result:

   ```
   Kernel driver in use: intel_fpga_pcie_drv
   ```

2.7. Running the Design Example Application

1. Navigate to `./software/user/example` under the design example directory.
2. Compile the design example application:

   ```
   $ make
   ```
3. Run the test:

   ```
   $ sudo ./intel_fpga_pcie_link_test
   ```

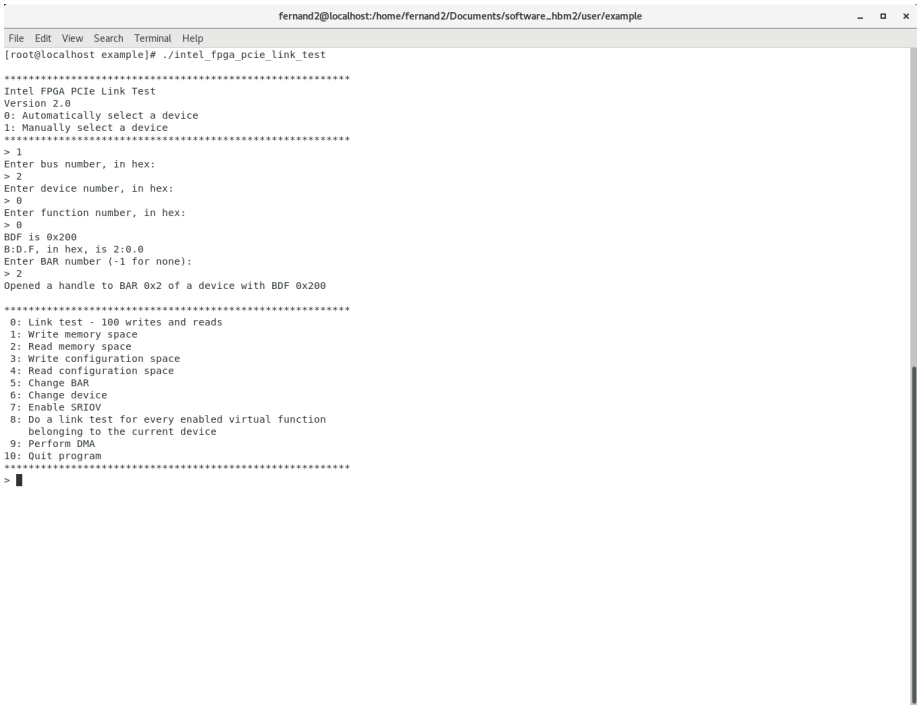
   You can run the Intel FPGA IP PCIe link test in manual or automatic mode.

   — In automatic mode, the application automatically selects the device. The test selects the Stratix 10 PCIe device with the lowest BDF by matching the Vendor ID. The test also selects the lowest available BAR.

   — In manual mode, the test queries you for the bus, device, and function number and BAR.

   For the Stratix 10-GX Development Kit, you can determine the BDF by typing the following command:

   ```
   $ lspci -d 1172
   ```
4. Here is a sample transcript for manual mode:



2.8. Ensuring the Design Example Meets Timing Requirements

If the generated design example fails to meet the timing requirements necessary to operate at 250 MHz, add pipeline registers to the design by performing the following steps:

1. Open the system in Platform Designer.
2. On the Menu bar, choose **System**, and then **Show System with Platform Designer Interconnect**. This opens another window.
3. In the new window, select the **Memory-Mapped Interconnect** tab. There is an option in the bottom left corner to **Show Pipelinable Locations**. Check that box. This shows locations where you can add pipeline registers.
4. Check your timing report to see where you need to add pipeline registers to achieve timing closure for your paths.
5. Right click where you want to add a pipeline register and check the **Pipelined** box. A pipeline register appears at that location. Repeat this step at all locations where you want to add pipeline registers.
6. Save and regenerate the system.
7. In the main window, click on the **Interconnect Requirements** tab. You can see the pipeline stages there.

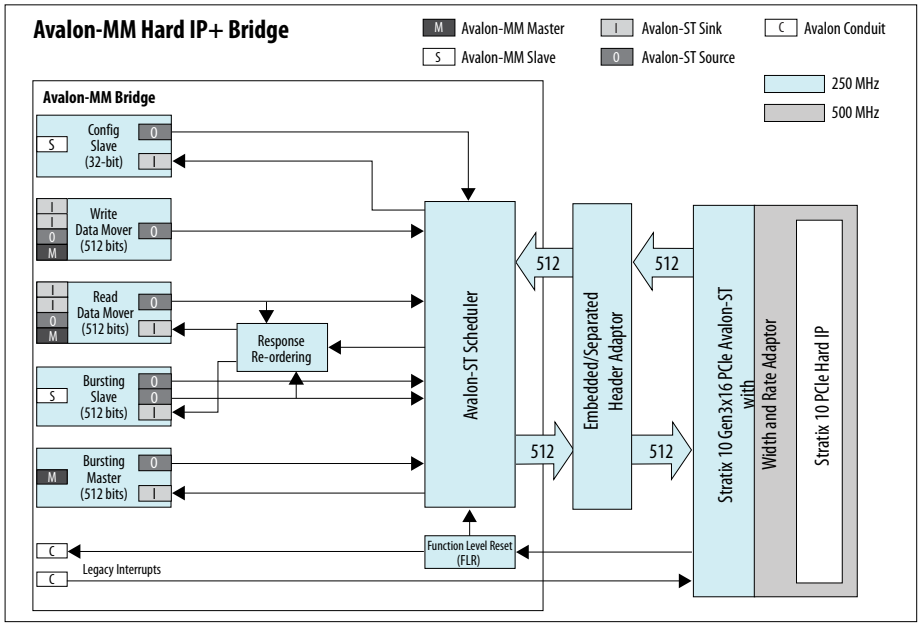
3. Block and Interface Descriptions

The Intel L-/H-Tile Avalon-MM+ for PCI Express IP consists of:

- Modules, implemented in soft logic, that perform Avalon-MM functions.
- A PCIe Hard IP that implements the Transaction, Data Link, and Physical layers required by the PCI Express protocol.

3.1. Block Descriptions

Figure 10. Top Level Block Diagram



Note: The Config Slave module is only present in Root Port variations.

The PCIe Hard IP implements the Transaction, Data Link, and Physical layers stack required by the PCI Express protocol. This stack allows the user application logic in the Stratix 10 FPGA to interface with another device via a PCI Express link.

While the PCIe Hard IP uses a 500 MHz clock, other modules in the Bridge use a 250 MHz clock. The Width and Rate Adapter handles the necessary conversion between these two clock domains. An additional adapter converts between the embedded header format of the PCIe Hard IP and the separate header and data format used by the other modules in the Avalon-MM Bridge.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

The Avalon-MM Bridge mainly consists of four leaf modules that interact with an Avalon-ST Scheduler. The four leaf modules are:

- Bursting Master: This module converts memory read and write TLPs received over the PCIe link into Avalon-MM burst read and write transactions, and sends back CplD TLPs for read requests it receives.
- Bursting Slave: This module converts Avalon-MM read and write transactions into PCIe memory read and write TLPs to be transmitted over the PCIe link. This module also processes the CplD TLPs received for the read requests it sent.
- Write Data Mover: This module uses PCIe memory write TLPs and Avalon-MM read transactions to move large amounts of data from your application logic in the Avalon-MM space to the system memory in the PCIe space.
- Read Data Mover: This module uses PCIe memory read TLPs and Avalon-MM write transactions to move large amounts of data from the system memory in the PCIe space to the FPGA memory in the Avalon-MM space.

The Avalon-ST Scheduler serves two main functions. In the receive direction, this module distributes the received TLPs to the appropriate leaf module based on the TLP's type, address and tag. In the transmit direction, this module takes TLPs ready for transmission from the leaf modules and schedules them for transfer to the Width and Rate Adapter to be forwarded to the PCIe Hard IP, based on the available credits. These data transfers follow the PCI Express Specifications' ordering rules to avoid deadlocks.

Descriptors provided to the Data Movers through one of their Avalon-ST sink interfaces control the data transfers. The Data Movers report the transfers' status through their Avalon-ST source interfaces.

The Config Slave converts single-cycle, 32-bit Avalon-MM read and write transactions into PCIe configuration read and write TLPs (CfgRd0, CfgRd1, CfgWr0 and CfgWr1) to be sent over the PCIe link. This module also processes the completion TLPs (Cpl and CplD) it receives in return.

Note: The Config Slave module is only present in Root Port variations. Root Port mode is a future enhancement, and is not supported in this release of the Quartus Prime Pro Edition.

The Response Reordering module assembles and reorders completion TLPs received over the PCIe link for the Bursting Slave and the Read Data Mover. It routes the completions based on their tag (see Tags for tag allocation information).

No re-ordering is necessary for the completions sent to the Config module as it only issues one request TLP at a time.

The FLR module detects Function-Level Resets received over the PCIe link and informs the application logic and the Scheduler module.

Note: The Function-Level Reset (FLR) capability is not currently available. It may be available in a future release of Quartus Prime.

You can individually enable or disable the Bursting Master and Slave, and the Read and Write Data Mover modules by making GUI selections in any combination in the **Parameters Editor** inside the Quartus Prime Pro Edition.

Endpoint applications typically need the Bursting Master to enable the host to provide information for the other modules.

3.1.1. Tags

The modules that issue non-posted requests need tags to associate the completions with the requests.

The 256 available tags are split into:

- 64 tags allocated to the Bursting Slave.
- 64 tags allocated to the Read Data Mover.

The remaining 128 tags are not used.

Note: The Config Slave module consumes one of these remaining 128 tags when Root Port mode is available.

The tag field of outgoing posted requests is a "don't care" and is set to 0.

3.2. Interface Descriptions

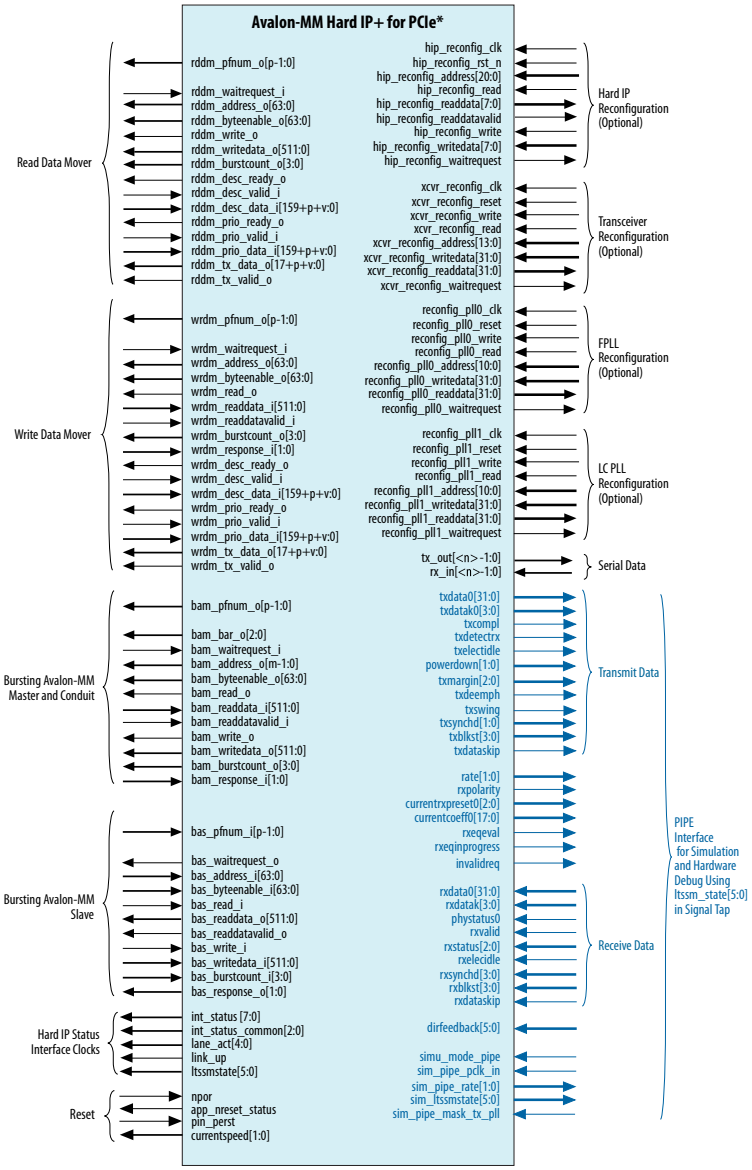
The Intel L-/H-Tile Avalon-MM+ for PCI Express IP includes many interface types to implement different functions.

These include:

- High performance bursting master and slave Avalon-MM interfaces to translate between PCIe TLPs and Avalon-MM memory-mapped reads and writes
- Read and Write Data Movers to transfer large blocks of data
- Standard PCIe serial interface to transfer data over the PCIe link
- System interfaces for interrupts, clocking, reset
- Optional reconfiguration interface to dynamically change the value of Configuration Space registers at run-time
- Optional status interface for debug

Unless otherwise noted, all interfaces to the Application layer are synchronous to the rising edge of the main system clock `coreclkout_hip` running at 250 MHz. The frequency of this clock is exactly half the frequency of the `hip_clk` generated by the Hard IP, with 0ppm difference. You enable the interfaces using the component GUI.

Figure 11. Intel L-/H-Tile Avalon-MM+ for PCI Express IP



Note: p = number of bits to address all the Functions.

m = Bursting Master address bus width.

s = Bursting Slave address bus width.

Read Data Mover (RDDM) interface: This interface transfers DMA data from the PCIe system memory to the memory in Avalon-MM address space.

Write Data Mover (WRDM) interface: This interface transfers DMA data from the memory in Avalon-MM address space to the PCIe system memory.

Bursting Master (BAM) interface: This interface provides host access to the registers and memory in Avalon-MM address space. The Busting Master module converts PCIe Memory Reads and Writes to Avalon-MM Reads and Writes.

Bursting Slave (BAS) interface: This interface allows the user application in the FPGA to access the PCIe system memory. The Bursting Slave module converts Avalon-MM Reads and Writes to PCIe Memory Reads and Writes.

The modular design of the Intel L-/H-Tile Avalon-MM+ for PCI Express IP lets you enable just the interfaces required for your application.

3.2.1. Avalon-MM Interface Summary

Table 9. Avalon-MM Interface Summary

Avalon-MM Type	Data Bus Width	Max Burst Size	Byte Enable Granularity	Max Outstanding Read Request
Bursting Slave	512 bits	8 cycles	byte	64
Bursting Master	512 bits	8 cycles	byte	32
Read Data Mover Write Master	512 bits	8 cycles	dword	N/A
Write Data Mover Read Master	512 bits	8 cycles	dword	128
Config Slave	32 bits	1 cycle	byte	1

Note: The number of read requests issued by the Write Data Mover's Avalon-MM Read Master is controlled by the assertion of `waitrequest` by the connected slave(s). The Read Master can handle 128 outstanding cycles of data. You cannot set this parameter in Platform Designer. The slave needs to correctly back-pressure the master once it cannot handle the incoming requests.

Note: The 512-bit Bursting Slave interface does not support transactions where all byte enables are set to 0.

3.2.1.1. Read Data Mover (RDDM) Interfaces

The Read Data Mover has four user-visible interfaces, described in the following sections.

3.2.1.1.1. Read Data Mover Avalon-MM Write Master and Conduit

Table 10. Read Data Mover Avalon-MM Write Master and Conduit Interface

Signal Name	Direction	Description
`rddm_pfnum_o[<PFNUM_WIDTH>-1:0]`	Output	Avalon conduit showing function number.
`rddm_waitrequest_i`	Input	Standard Avalon-MM Write Master interface. For details, refer to the *Avalon Interface Specifications*. WaitRequest Allowance parameter for this interface is 16.
`rddm_write_o`	Output	

continued...

Signal Name	Direction	Description
rddm_address_o[63:0]	Output	
rddm_burstcount_o[3:0]	Output	
rddm_byteenable_o[63:0]	Output	
rddm_writedata_o[511:0]	Output	

Note: PFNUM_WIDTH is the logarithm in base 2 of the number of Physical Functions (rounded up to the next integer).

Related Information

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

3.2.1.1.2. Read Data Mover Avalon-ST Descriptor Sinks

The Read Data Mover has two Avalon-ST sinks through which it receives the descriptors that define the data transfers. One of the interfaces receives descriptors for normal data transfers, while the other receives descriptors for high-priority data transfers.

Table 11. Read Data Mover normal descriptor sink interface

Signal Name	Direction	Description
rddm_desc_ready_o	Output	When asserted, this ready signal indicates the normal descriptor queue in the Read Data Mover is ready to accept data. The ready latency of this interface is 3 cycles.
rddm_desc_valid_i	Input	When asserted, this signal qualifies valid data on any cycle where data is being transferred to the normal descriptor queue. On each cycle where this signal is active, the queue samples the data.
rddm_desc_data_i[159:0]	Input	[159:152] : descriptor ID [151:149] : application specific [148] : single destination [147] : reserved [146] : reserved [145:128] : number of dwords to transfer up to 1 MB [127:64] : destination Avalon-MM address [63:0] : source PCIe address

When the single destination bit is set, the transfers use the same destination address. If the bit is not set, the address increments for each transfer.

Note: When the single source bit is set, the Avalon-MM destination address and the PCIe source address must be a multiple of 64.

Table 12. Read Data Mover priority descriptor sink interface

Signal Name	Direction	Description
rddm_prio_ready_o	Output	When asserted, this ready signal indicates the priority descriptor queue in the Read Data Mover is ready to accept data.This ready latency of this interface is 3 cycles.
rddm_prio_valid_i	Input	When asserted, this signal qualifies valid data on any cycle where data is being transferred to the priority descriptor queue. On each cycle where this signal is active, the queue samples the data.
rddm_prio_data_i[159:0]	Input	[159:152] : descriptor ID [151:149] : application specific [148] : single destination [147] : reserved [146] : reserved [145:128] : number of dwords to transfer up to 1 MB [127:64] : destination Avalon-MM address [63:0] : source PCIe address

The Read Data Mover internally keeps two queues of descriptors. The priority queue has absolute priority over the normal queue. Use it carefully to avoid starving the normal queue.

If the Read Data Mover receives a descriptor on the priority interface while processing a descriptor from the normal queue, it switches to processing descriptors from the priority queue as soon as it has completed the current descriptor. The Read Data Mover resumes processing the descriptors from the normal queue once the priority queue is empty. Do not use the same descriptor ID simultaneously in the two queues as there would be no way to distinguish them on the Status Avalon-ST source interface.

3.2.1.1.3. Read Data Mover Status Avalon-ST Source

Table 13. Read Data Mover Status Avalon-ST Source Interface

Signal Name	Direction	Description
rddm_tx_data_o[18+<*PFNUM_WIDTH*>-1:18]	Output	These bits contain the function number.
rddm_tx_data_o[17:0]	Output	[17:16] : reserved [15] : reserved *Note:* When the **Enable Completion Checking** option in the **Avalon-MM Settings** tab of the GUI is **On**, this bit becomes the completion timeout error flag. [14:12] : application specific [11:9] : reserved [8] : priority [7:0] : descriptor ID
rddm_tx_valid_o	Output	Valid status signal.

This interface does not have a ready input. The application logic must always be ready to receive status information for any descriptor that it has sent to the Read Data Mover.

The Read Data Mover copies over the application specific bits in the `rddm_tx_data_o` bus from the corresponding descriptor. A set priority bit indicates that the descriptor is from the priority descriptor sink.

A status word is output on this interface when the processing of a descriptor has completed, including the reception of all completions for all memory read requests.

3.2.1.2. Write Data Mover (WRDM) Interfaces

The Write Data Mover has four user-visible interfaces, described in the following sections.

3.2.1.2.1. Write Data Mover Avalon-MM Read Master and Conduit

Table 14. Write Data Mover Avalon-MM Read Master and Conduit Interface

Signal Name	Direction	Description
`wrdm_pfnum_o[<PFNUM_WIDTH>-1:0]`	Output	Avalon conduit showing function number.
`wrdm_waitrequest_i`	Input	Standard Avalon-MM Read Master interface. For details, refer to the *AvalonInterface Specifications*. Byte enables are all ones for bursts larger than one cycle.
`wrdm_read_o`	Output	
`wrdm_address_o[63:0]`	Output	
`wrdm_burstcount_o[3:0]`	Output	
`wrdm_byteenable_o[63:0]`	Output	
`wrdm_readdatavalid_i`	Input	
`wrdm_readdata_i[511:0]`	Input	
`wrdm_response_i[1:0]`	Input	

The waitrequestAllowance of this interface is four.

Related Information

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

3.2.1.2.2. Write Data Mover Avalon-ST Descriptor Sinks

Table 15. Write Data Mover normal descriptor sink interface

Signal Name	Direction	Description
`wrdm_desc_ready_o`	Output	When asserted, this ready signal indicates the normal descriptor queue in the Write Data Mover is ready to accept data. The readyLatency is 3.
`wrdm_desc_valid_i`	Input	When asserted, this signal qualifies valid data on any cycle where data is being transferred to the normal

continued...

Signal Name	Direction	Description
		descriptor queue. On each cycle where this signal is active, the queue samples the data.
wrdm_desc_data_i[159:0]	Input	[159:152] : descriptor ID [151:149] : application specific [148] : reserved [147] : single source [146] : immediate [145:128] : number of dwords to transfer up to 1 MB [127:64] : destination PCIe address [63:0] : source Avalon-MM address / immediate data

When the single source bit is set, the source address is used for all the transfers unchanged. If the bit is not set, the address increments for each transfer

Note: When the single source bit is set, the Avalon-MM source address and the PCIe destination address must be a multiple of 64.

When set, the immediate bit indicates immediate writes. The Write Data Mover supports immediate writes of one or two dwords. For immediate transfers, bits [31:0] or [63:0] contain the payload for one or two dwords transfers respectively. The two dwords immediate writes cannot cross a four KB boundary.

Table 16. Write Data Mover priority descriptor sink interface

Signal Name	Direction	Description
wrdm_prio_ready_o	Output	When asserted, this ready signal indicates the priority descriptor queue in the Write Data Mover is ready to accept data. This readyLatency is 3 cycles.
wrdm_prio_valid_i	Input	When asserted, this signal qualifies valid data on any cycle where data is being transferred to the priority descriptor queue. On each cycle where this signal is active, the queue samples the data.
wrdm_prio_data_i[159:0]	Input	[159:152] : descriptor ID [151:149] : application specific [148] : reserved [147] : single source [146] : immediate [145:128] : number of dwords to transfer up to 1 MB [127:64] : destination PCIe address [63:0] : source Avalon-MM address / immediate data

The Write Data Mover internally keeps two queues of descriptors. The priority queue has absolute priority over the normal queue. Use it carefully to avoid starving the normal queue.

If the Write Data Mover receives a descriptor on the priority interface while processing a descriptor from the normal queue, it switches to processing descriptors from the priority queue after it has completed processing the current descriptor from the normal queue. The Write Data Mover resumes processing descriptors from the normal queue once the priority queue is empty. Do not use the same descriptor ID simultaneously in the two queues as there would be no way to distinguish them on the Status Avalon-ST source interface.

3.2.1.2.3. Write Data Mover Status Avalon-ST Source

Table 17. Write Data Mover Status Avalon-ST Source Interface

Signal Name	Direction	Description
wrdm_tx_data_o[18+<*PFNUM_WIDTH*>-1:18]	Output	These bits contain the function number.
wrdm_tx_data_o[17:0]	Output	[17:16] : reserved [15] : error [14:12] : application specific [11:9] : reserved [8] : priority [7:0] : descriptor ID
wrdm_tx_valid_o	Output	Valid status signal.

This interface does not have a ready input. The application logic must always be ready to receive status information for any descriptor that it has sent to the Write Data Mover.

The ready latency does not matter because there is no ready input.

The Write Data Mover copies over the application specific bits in the wrdm_tx_data_o bus from the corresponding descriptor. A set priority bit indicates that the descriptor was from the priority descriptor sink.

3.2.1.3. Bursting Avalon-MM Slave (BAS) Interface

The Bursting Avalon-MM Slave module has one user-visible Avalon-MM slave interface.

For more details on these interface signals, refer to the *Avalon Interface Specifications*.

Table 18. Bursting Slave Avalon-MM Slave and Conduit Interface

Signal Name	Direction	Description
bas_pfnum_i[<*PFNUM_WIDTH*>-1:0]	Input	Avalon conduit showing function number.
bas_waitrequest_o	Output	This signal provides a back-pressure mechanism. It forces the master to wait until the system interconnect fabric is ready to proceed with the transfer.
bas_address_i[63:0]	Input	
bas_byteenable_i[63:0]	Input	The BAS interface supports all contiguous byteenable patterns. For burst read transactions, the byte enables must be 64'hFFFF_FFFF_FFFF_FFFF.

continued...

Signal Name	Direction	Description
bas_read_i	Input	
bas_readdata_o[511:0]	Output	
bas_readdatavalid_o	Output	
bas_response_o[1:0]	Output	These bits contain the response status for any transaction happening on the BAS interface: • 00: OKAY - Successful response for a transaction. • 01: RESERVED - Encoding is reserved. • 10: SLAVEERROR - Error from an endpoint slave. Indicates an unsuccessful transaction. • 11: DECODEERROR - Indicates attempted access to an undefined location.
bas_write_i	Input	
bas_writedata_i[511:0]	Input	
bas_burstcount_i[3:0]	Input	

Table 19. Bursting Slave Avalon-MM Slave Parameters

Name	Value
waitrequestAllowance	0
maximumPendingReadTransactions	64

Related Information

Avalon Interface Specifications

3.2.1.4. Bursting Avalon-MM Master (BAM) Interface

The Bursting Avalon-MM Master module has one user-visible Avalon-MM Master interface.

The bam_bar_o bus contains the BAR address for a particular TLP. This bus is as an extension of the standard address bus.

Table 20. Bursting Master Avalon-MM Master and Conduit Interface

Signal Name	Direction	Description
bam_pfnum_o[PFNUM_WIDTH-1:0]	Output	Avalon conduit showing function number.
bam_bar_o[2:0]	Output	000 : Memory BAR 0 001 : Memory BAR 1 010 : Memory BAR 2 011 : Memory BAR 3 100 : Memory BAR 4 101 : Memory BAR 5 110 : Reserved 111 : Expansion ROM BAR
bam_waitrequest_i	Input	Avalon-MM wait request signal with waitrequestAllowance of 8.

continued...

Signal Name	Direction	Description
bam_address_o[BAM_ADDR_WIDTH-1:0]	Output	The width of the Bursting Master's address bus is the maximum of the widths of all the enabled BARs. For BARs narrower than the widest BAR, the address bus' additional most-significant bits are driven to 0.
bam_byteenable_o[63:0]	Output	The BAM interface supports all contiguous byteenable patterns.
bam_read_o	Output	
bam_readdata_i[511:0]	Input	
bam_readdatavalid_i	Input	
bam_response_i[1:0]	Input	Reserved. Drive these inputs to 0.
bam_write_o	Output	
bam_writedata_o[511:0]	Output	
bam_burstcount_o[3:0]	Output	

For more details on these interface signals, refer to the *Avalon Interface Specifications*.

Related Information

Avalon Interface Specifications

3.2.1.4.1. PCIe Address to Avalon-MM Address Mapping

The Bursting Master module transforms PCIe memory read (MRd) and memory write (MWr) request packets received from the PCIe system into Avalon-MM read and write transactions.

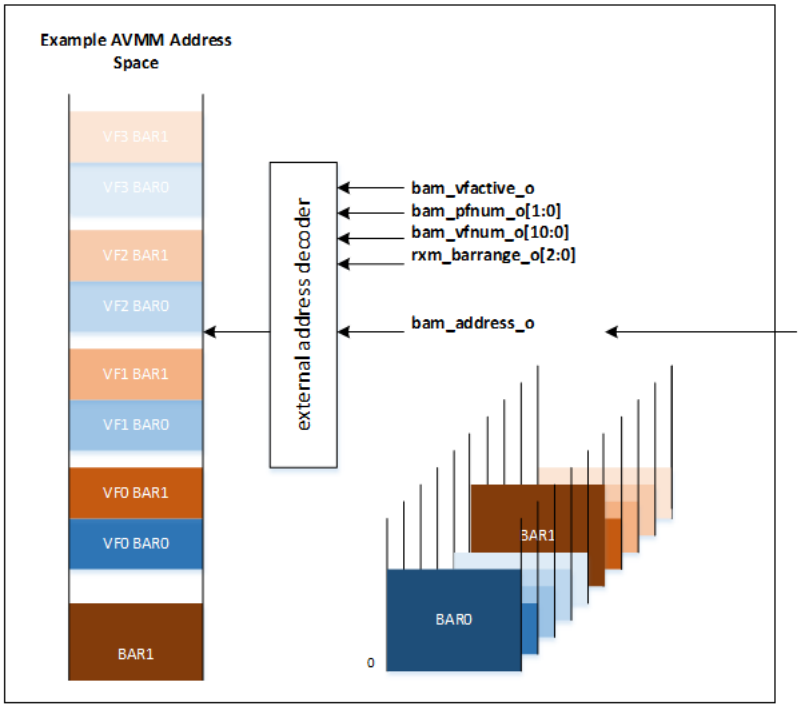
The Bursting Master module compares the address of each incoming read or write TLP with the BARs defined in the PCIe Hard IP. For TLPs that match with a BAR, the offset from the BAR is put out on the address bus and the number of the matching BAR comes out on the `bam_bar_o` bus.

Although these signals are in conduits separate from the Avalon-MM master interface, they are synchronous to it and can be treated as extensions of the address bus. The user logic can decode the Avalon-MM address and physical/virtual function numbers to translate them into the Avalon-MM address space.

Because the additional signals are in a separate conduit, they must be routed along with the signals from the Avalon-MM master interface to a single module with an Avalon-MM slave interface and matching conduits for the BAR and function numbers. This module decodes the BAR information contained in the `bam_bar_o` signals in the conduit, and routes the data to one or more standard Avalon-MM slaves. This decoder (referred to as the BAR Interpreter in the available design example) is necessary because Platform Designer cannot route the conduit to multiple points.

The following figure shows an example of translating the PCIe address into an Avalon-MM address. This example shows that PCIe address regions grouped with BAR number are translated into Avalon-MM address regions grouped with virtual function number.

Figure 12. Example PCIe to Avalon-MM Address Mapping



3.2.2. Clocks and Resets

The Intel L-/H-Tile Avalon-MM+ for PCI Express IP generates the 250 MHz Application clock, `coreclkout_hip` and the `reset` signal. This IP core also provides a synchronized version of the `reset` signal, `app_nreset_status`, to the Application. This is an active low reset.

Figure 13. Intel L-/H-Tile Avalon-MM+ for PCI Express IP Clock and Reset Connections

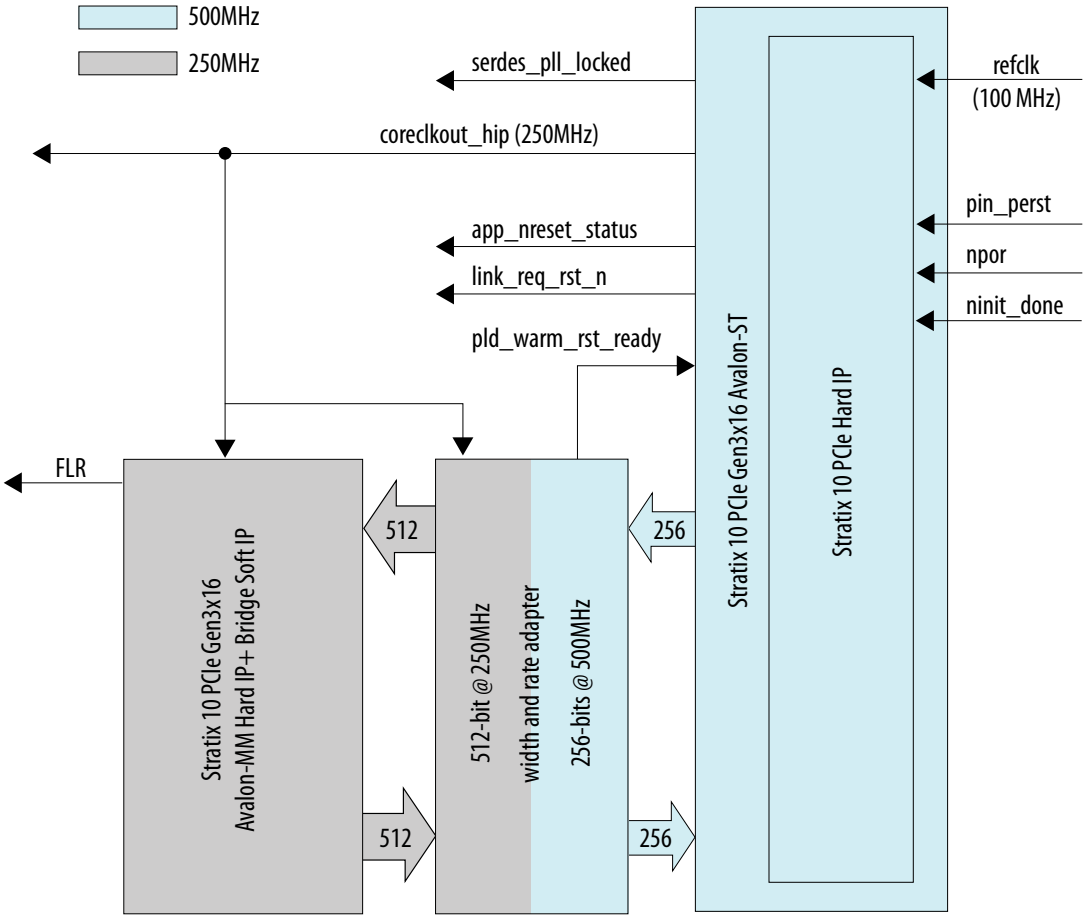


Table 21. Application Layer Clock Frequency

Link Width	Maximum Link Rate	Avalon Interface Width	coreclkout_hip
×16	Gen3	512	250 MHz

Table 22. Clocks

Signal Name	Direction	Description
refclk	Input	100 MHz reference clock defined by the PCIe specification. To meet the PCIe 100ms wake-up time requirement, this clock must be free-running. *Note:* This input reference clock must be stable and free-running at device power-up for a successful device configuration.
serdes_pll_locked	Output	Asserted when coreclkout_hip is stable.
coreclkout_hip	Output	250 MHz application clock generated internally to the Intel L-/H-Tile Avalon-MM+ for PCI Express IP. If your

continued...

Signal Name	Direction	Description
		application logic does not use `coreclkout_hip` as the clock, you must insert clock-crossing logic between your application logic and the Intel L-/H-Tile Avalon-MM+ for PCI Express IP.

Table 23. Resets

Signal Name	Direction	Clock	Description
`pin_perst`	Input	Asynchronous	This is an active-low input to the PCIe Hard IP, and implements the PERST# function defined by the PCIe specification.
`npor`	Input	Asynchronous, edge-sensitive	This active-low warm reset signal is an input to the PCIe Hard IP, and resets the entire PCIe Hard IP. It should be held for a minimum of 20 ns. This signal is edge-sensitive, not level-sensitive.
`app_nreset_status`	Output	app_clk	This active-low signal is held high until the PCIe Hard IP is ready. It is only deasserted after `npor` and `pin_perst` are deasserted and the PCIe Hard IP has come out of reset.
`link_req_rst_n`	Output	hip_clk	The PCIe Hard IP asserts this active-low signal when it is about to go into reset. When this signal is asserted, the Intel L-/H-Tile Avalon-MM+ for PCI Express IP resets all its PCIe-related registers and queues, including anything related to tags. It stops sending packets to the PCIe Hard IP until the Bus Master Enable bit is set again, and ignores any packet received from the PCIe Hard IP.
`ninit_done`	Input	Asynchronous	A "1" on this active-low signal indicates that the FPGA device is not yet fully configured. A "0" indicates the device has been configured and is in normal operating mode.

Note: To use the `ninit_done` input, instantiate the Reset Release Intel FPGA IP in your design and use its `ninit_done` output to drive the input of the Intel L-/H-Tile Avalon-MM+ for PCI Express IP. For more details on how you can use the `ninit_done` signal, refer to *Stratix 10 Configuration User Guide*.

Related Information

Stratix 10 Configuration User Guide

3.2.3. System Interfaces

These system interfaces allow this IP core to communicate to the customer's application, and to the link partner via the PCIe link.

TX and RX Serial Data

This differential, serial interface is the physical link between a Root Port and an Endpoint. This IP Core supports 16 lanes, and operates in Gen3 at 8 GT/s. Each lane includes a TX and RX differential pair. Data is striped across all available lanes.

PIPE

This is a parallel interface between the PCIe IP Core and PHY. The PIPE data bus is 32 bits. Each lane includes four control/data bits and other signals. It carries the TLP data before it is serialized. It is available for simulation only and provides more visibility for debugging.

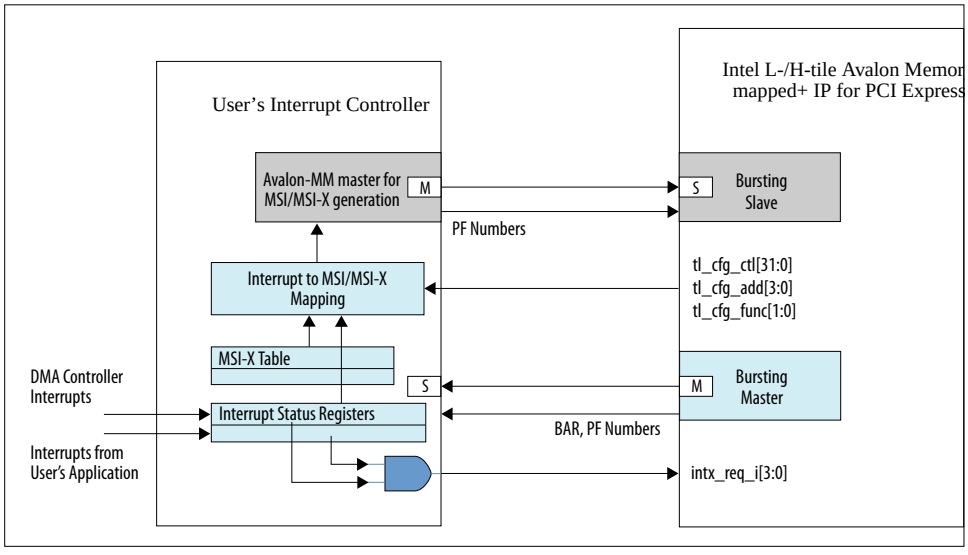
Interrupts

Legacy interrupts, MSI, and MSI-X interrupts are all controlled and generated externally to the Intel L-/H-Tile Avalon-MM+ for PCI Express IP to ensure total flexibility in allocating interrupt resources based on the needs of the application.

The host can initiate an MSI by including a descriptor for an immediate write at the end of the descriptor table that it places in system memory. When the Write Data Mover receives this descriptor on one of its descriptor queues, it can perform the memory write to send the MSI because the necessary MSI information (address and data) is contained in the descriptor itself.

The user application logic in the Avalon-MM space can also initiate an MSI by leveraging the Bursting Avalon Slave (BAS) interface.

Figure 14. Example of Interrupt Controller Integration with Endpoint Intel L-/H-Tile Avalon-MM+ for PCI Express IP



Hard IP Reconfiguration

This optional Avalon-MM interface allows you to dynamically update the value of read-only configuration registers at run-time. It is available when **Enable HIP dynamic reconfiguration of PCIe read-only registers** is enabled in the component GUI.

Hard IP Status and Link Training

This optional interface includes the following signals that are useful for debugging

- Link status signals
- Interrupt status signals
- TX and RX parity error signals
- Correctable and uncorrectable error signals

Related Information

PCI Express Base Specification 3.0

3.2.3.1. Serial Data Interface

This IP core supports 16 lanes.

Table 24. Serial Data Interface

Signal	Direction	Description
`tx_out[15:0]`	Output	Transmit serial data output.
`rx_in[15:0]`	Input	Receive serial data input.

3.2.3.2. PIPE Interface

Refer to the **Stratix 10 Avalon-ST and Single Root I/O Virtualization (SR-IOV) Interfaces for PCIe Solutions User Guide** for a description of the signals on the PIPE interface.

Note: Use this interface only in simulations.

3.2.3.3. Interrupts

The Intel L-/H-Tile Avalon-MM+ for PCI Express IP does not use the interrupt-related signals described in the next section. However, the user application logic can use them to generate or respond to interrupts.

3.2.3.3.1. Interrupt Signals Available when the PCIe Hard IP is an Endpoint

Table 25. Interrupt Signals when PCIe Hard IP is an Endpoint

Signal	Direction	Description
The following signals are in the `hip_clk` clock domain		
`intx_req_i[3:0]`	Input	The Bridge IP core exports these legacy interrupt request signals from the PCIe Hard IP directly to the Application Layer interface. When these signals go high, they indicate that an assertion of the corresponding INTx messages are requested. When they go low, they indicate that a deassertion of the corresponding INTx messages are requested. These signals are only present when legacy interrupts are enabled.

3.2.3.3.2. MSI

MSI interrupts are signaled on the PCI Express link using Memory Write (MWr) TLPs. You can obtain the necessary MSI information (such as the message address and data) from the configuration output interface (`tl_cfg_*`) to create the MWr TLPs.

3.2.3.4. Configuration Output Interface

The Transaction Layer (TL) bus provides a subset of the information stored in the Configuration Space. Use this information in conjunction with the `app_err*` signals to understand TLP transmission problems.

Table 26. Configuration Output Interface Signals

Signal	Direction	Description
`tl_cfg_add[3:0]` (H-tile) `tl_cfg_add[4:0]` (L-tile)	Output	Address of the TLP register. This signal is an index indicating which Configuration Space register information is being driven onto `tl_cfg_ctl[31:0]`.
`tl_cfg_ctl[31:0]`	Output	The `tl_cfg_ctl` signal is multiplexed and contains a subset of contents of the Configuration Space registers.
`tl_cfg_func[1:0]`	Output	Specifies the function whose Configuration Space register values are being driven onto `tl_cfg_ctl`. The following encodings are defined: • 2'b00: Physical Function (PF0) • 2'b01: PF1 for H-tile, reserved for L-tile • 2'b10: PF2 for H-tile, reserved for L-tile • 2'b11: PF3 for H-tile, reserved for L-tile

Information on the `tl_cfg_ctl` bus is time-division multiplexed (TDM). Examples of information multiplexed onto the `tl_cfg_ctl` bus include device number, bus number, MSI information (address, data, mask) and AER information. For more details, refer to the *Transaction Layer Configuration Space Interface* section of the Stratix 10 Avalon streaming and Single Root I/O Virtualization (SR-IOV) Interface for PCI Express Solutions User Guide.

Note: In the 20.3 release of Quartus Prime, the configuration output interface (`tl_cfg_*`) is exported by default.

Related Information

Stratix 10 Avalon streaming and Single Root I/O Virtualization (SR-IOV) Interface for PCI Express Solutions User Guide

3.2.3.5. Flush Requests

In the PCI Express protocol, a memory read request from the host with a length of 1 dword and byte enables being all 0's translate to a flush request for the Completer, which in this case is the Intel L-/H-Tile Avalon-MM+ for PCI Express IP. However, this flush request feature is not supported by the Intel L-/H-Tile Avalon-MM+ for PCI Express IP.

3.2.3.6. Reconfiguration

3.2.3.6.1. PCIe Hard IP Reconfiguration

This interface is a way to access the PCIe Hard IP's internal registers and the PCIe configuration registers. It has its own clock and reset signals.

For more details on the signals in this interface, refer to the *Stratix 10 Avalon-MM Interface for PCI Express Solutions User Guide*.

Table 27. PCIe HIP Reconfiguration Interface Signals

Signal Name	Direction	Description
hip_reconfig_clk	Input	PCIe Hard IP reconfiguration clock
hip_reconfig_reset	Input	PCIe Hard IP reconfiguration reset
hip_reconfig_write	Input	Standard Avalon-MM interface. For details, refer to the *Avalon Interface Specifications*.
hip_reconfig_read	Input	
hip_reconfig_address[20:0]	Input	
hip_reconfig_writedata[7:0]	Input	
hip_reconfig_readdata[7:0]	Output	
hip_reconfig_waitrequest	Output	

Related Information

- Avalon Interface Specifications
  For information about the Avalon-MM interfaces to implement read and write interfaces for master and slave components.
- Stratix 10 Avalon-MM Interface for PCI Express Solutions User Guide
  For information about the signals in the Hard IP Reconfiguration interface..

3.2.3.6.2. Transceiver Reconfiguration

This is an optional interface that users can enable by selecting the **Enable Transceiver Dynamic Reconfiguration** option from the GUI.

Table 28. Transceiver Reconfiguration Interface

Signal Name	Direction	Description
xcvr_reconfig_clk	Input	Transceiver reconfiguration clock.
xcvr_reconfig_reset	Input	Transceiver reconfiguration reset.
xcvr_reconfig_write	Input	Standard Avalon-MM interface. For details, refer to the **Avalon Interface Specifications**.
xcvr_reconfig_read	Input	
xcvr_reconfig_address[31:0]	Input	
xcvr_reconfig_writedata[31:0]	Input	
xcvr_reconfig_readdata[31:0]	Output	
xcvr_reconfig_waitrequest	Output	

3.2.3.6.3. PLL Reconfiguration

There are two interfaces for PLL reconfiguration. One interface is for reconfiguring the FPLL, and the other is for reconfiguring the LC PLL.

Table 29. FPLL Reconfiguration Interface

Signal Name	Direction	Description
reconfig_pll0_clk	Input	FPLL reconfiguration clock.
reconfig_pll0_reset	Input	FPLL reconfiguration reset.
reconfig_pll0_write	Input	Standard Avalon-MM interface. For details, refer to the **Avalon Interface Specifications**.
reconfig_pll0_read	Input	
reconfig_pll0_address[10:0]	Input	
reconfig_pll0_writedata[31:0]	Input	
reconfig_pll0_readdata[31:0]	Output	
reconfig_pll0_waitrequest	Output	

Table 30. LC PLL Reconfiguration Interface

Signal Name	Direction	Description
reconfig_pll1_clk	Input	LC PLL reconfiguration clock.
reconfig_pll1_reset	Input	LC PLL reconfiguration reset.
reconfig_pll1_write	Input	Standard Avalon-MM interface. For details, refer to the **Avalon Interface Specifications**.
reconfig_pll1_read	Input	
reconfig_pll1_address[10:0]	Input	
reconfig_pll1_writedata[31:0]	Input	
reconfig_pll1_readdata[31:0]	Output	
reconfig_pll1_waitrequest	Output	

3.2.3.7. Hard IP Status and Link Training Conduit

The following signals are provided for debug and monitoring purposes. They come directly from the PCIe HIP, and are in the `hip_clk` clock domain.

Table 31. PCIe HIP Status and Link Training Conduit Signals

Signal Name	Direction	Description
link_up_o	Output	When asserted, this signal indicates that the link is up.
ltssmstate_o[5:0]	Output	These are the state variables for the LTSSM state machine. Their encoding defines the following states: 6'h00 : DETECT_QUIET 6'h01 : DETECT_ACT 6'h02 : POLL_ACTIVE 6'h03 : POLL_COMPLIANCE 6'h04 : POLL_CONFIG 6'h05 : PRE_DETECT_QUIET

continued...

Signal Name	Direction	Description
		6'h06 : DETECT_WAIT 6'h07 : CFG_LINKWD_START 6'h08 : CFG_LINKWD_ACCEPT 6'h09 : CFG_LANENUM_WAIT 6'h0A : CFG_LANENUM_ACCEPT 6'h0B : CFG_COMPLETE 6'h0C : CFG_IDLE 6'h0D : RCVRY_LOCK 6'h0E : RCVRY_SPEED 6'h0F : RCVRY_RCVRCFG 6'h10 : RCVRY_IDLE 6'h20 : RCVRY_EQ0 6'h21 : RCVRY_EQ1 6'h22 : RCVRY_EQ2 6'h23 : RCVRY_EQ3 6'h11 : L0 6'h12 : L0S 6'h13 : L123_SEND_EIDLE 6'h14 : L1_IDLE 6'h15 : L2_IDLE 6'h16 : L2_WAKE 6'h17 : DISABLED_ENTRY 6'h18 : DISABLED_IDLE 6'h19 : DISABLED 6'h1A : LPBK_ENTRY 6'h1B : LPBK_ACTIVE 6'h1C : LPBK_EXIT 6'h1D : LPBK_EXIT_TIMEOUT 6'h1E : HOT_RESET_ENTRY 6'h1F : HOT_RESET
currentspeed_o[1:0]	Output	These signals indicate the current speed of the PCIe link. The following encodings are defined: 2'b00 : Undefined 2'b01 : Gen1 2'b10 : Gen2 2'b11 : Gen3
lane_act_o[4:0]	Output	These signals indicate the number of lanes that are configured during link training. The following encodings are defined: 5'b0 0001 : 1 lane 5'b0 0010 : 2 lanes 5'b0 0100 : 4 lanes 5'b0 1000 : 8 lanes 5'b1 0000 : 16 lanes
int_status[10:0] (H-Tile) int_status[7:0] (L-Tile)	Output	The `int_status[3:0]` signals drive legacy interrupts to the application. • `int_status[0]`: Interrupt signal A • `int_status[1]`: Interrupt signal B • `int_status[2]`: Interrupt signal C • `int_status[3]`: Interrupt signal D The `int_status[10:4]` signals provide status for other interrupts.

continued...

Signal Name	Direction	Description
		• int_status[4]: Specifies a Root Port AER error interrupt. This bit is set when the cfg_aer_rc_err_msi or cfg_aer_rc_err_int signal asserts. This bit is cleared when software writes 1 to the register bit or when cfg_aer_rc_err_int is deasserted. • int_status[5]: Specifies the Root Port PME interrupt status. It is set when cfg_pme_msi or cfg_pme_int asserts. It is cleared when software writes a 1 to clear or when cfg_pme_int deasserts. • int_status[6]: Is asserted when a hot plug event occurs and Power Management Events (PME) are enabled. (PMEs are typically used to revive the system or a function from a low power state). • int_status[7]: Specifies the hot plug event interrupt status. • int_status[8]: Specifies the interrupt status for the Link Autonomous Bandwidth Status register. H-Tile only. • int_status[9]: Specifies the interrupt status for the Link Bandwidth Management Status register. H-Tile only. • int_status[10]: Specifies the interrupt status for the Link Equalization Request bit in the Link Status register. H-Tile only.
int_status_common[2:0]	Output	Specify the interrupt status for the following registers. When asserted, each signal indicates that an interrupt is pending: • int_status_common[0]: Autonomous bandwidth status register. • int_status_common[1]: Bandwidth management status register. • int_status_common[2]: Link equalization request bit in the link status register.

3.2.3.8. Bus Master Enable (BME) Conduit

Table 32. BME Conduit Signals

Signal Name	Direction	Description
bus_master_enable_o[3:0]	Output	Indicate the status of the bus_master_enable bit (i.e bit 2) in the PCI command register. This bit specifies if a function is capable of issuing Memory and IO Read/Write requests. Refer to the PCIe Specifications for more details. • bus_master_enable_o[0] corresponds to function 0. • bus_master_enable_o[1] corresponds to function 1. • bus_master_enable_o[2] corresponds to function 2. • bus_master_enable_o[3] corresponds to function 3.

3.2.3.9. Test Conduit

The Intel L-/H-Tile Avalon-MM+ for PCI Express IP does not export the signals in this conduit, which comes from the PCIe Hard IP.

Unused inputs must be tied low.

Table 33. Test Conduit Signals

Signal Name	Direction	Description
test_in_i[66:0]	Input	For debugging purpose. These signals control what internal signals are brought out on the test_out_o bus.
aux_test_out_o[6:0]	Output	For debugging purpose.
test_out_o[255:0]	Output	For debugging purpose.
sim_pipe_mask_tx_pll_lock_i	Input	This signal is used only in BFM PIPE simulation.

4. Parameters

This chapter provides a reference for all the parameters of the Intel L-/H-Tile Avalon-MM+ for PCI Express IP core.

Table 34. Design Environment Parameter

Starting in 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 35. System Settings

Parameter	Value	Description
Application Interface Type	**Avalon-MM**	By default, the interface to the Application Layer is set to Avalon-MM.
Hard IP Mode		By default, the Hard IP mode is set to Gen3 x16, with a 512-bit interface to the Application Layer running at 250 MHz.
Port type	**Native Endpoint** **Root Port**	Specifies the port type. The Endpoint stores parameters in the Type 0 Configuration Space. The Root Port stores parameters in the Type 1 Configuration Space. *Note:* Root Port mode is not available in the 18.0 release of Quartus Prime Pro Edition.

4.1. Avalon-MM Settings

Table 36. Avalon-MM Settings

Parameter	Value	Default	Description
Enable Read Data Mover	**On/Off**	**On**	When **On**, the IP core includes the Read Data Mover.
Address width of Read Data Mover	**10 - 64 bits**	**64 bits**	Specifies the address width for the Read Data Mover.
Enable Write Data Mover	**On/Off**	**On**	When **On**, the IP core includes the Write Data Mover.
Address width of Write Data Mover	**10 - 64 bits**	**64 bits**	Specifies the address width for the Write Data Mover.

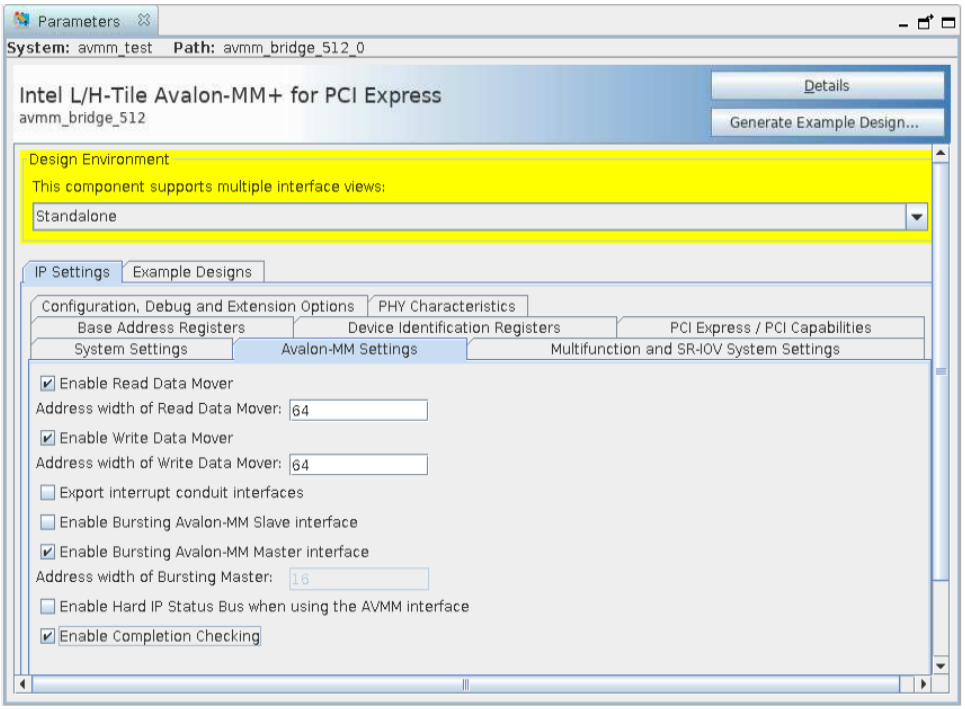
continued...

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Parameter	Value	Default	Description
Export interrupt conduit interfaces	**On/Off**	**Off**	When **On**, the IP core exports internal interrupt signals to the top-level RTL module. The exported signals support legacy interrupts.
Enable bursting Avalon-MM Slave interface	**On/Off**	**Off**	When **On**, the high performance Avalon-MM Slave interface is enabled. This interface is appropriate for high bandwidth applications such as transferring blocks of data. The maximum burst count supported is 8.
Address width of bursting Slave	**10 - 64 bits**	**64 bits**	Specifies the address width for the high performance Avalon-MM Slave interface. This parameter is accessible only after the high performance bursting Avalon-MM Slave interface is enabled.
Enable bursting Avalon-MM Master interface	**On/Off**	**On**	When **On**, the high performance Avalon-MM Master interface is enabled. This interface is appropriate for high bandwidth applications such as transferring blocks of data. The maximum burst count supported is 8.
Enable hard IP status bus when using the Avalon-MM interface	**On/Off**	**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
Enable Completion Checking	**On/Off**	**Off**	With this option **On**, when the Read Data Mover in the IP core issues an upstream Read to the host and the host does not return a completion after a time period exceeding the completion timeout value specified in a PCIe Hard IP Configuration register, the output `rddm_tx_data_o[15]` is asserted to indicate a completion timeout error. No associated data is returned to the Read Data Mover for the descriptor that corresponds to the completion timeout error. If an external Avalon-MM Master issues a Read request to the BAS interface (which the BAS interface then forwards to the host), and a completion timeout happens, the BAS interface will respond to the Avalon-MM Master with dummy data and set the outputs `bas_response_o[1:0]` to 2'b10 to indicate an unsuccessful completion. The possible encodings for `bas_response_o[1:0]` are: • 00: OKAY - Successful response for a transaction. • 01: RESERVED - Encoding is reserved. • 10: SLAVEERROR - Error from an endpoint slave. Indicates an unsuccessful transaction. • 11: DECODEERROR - Indicates attempted access to an undefined location.

Figure 15. Avalon-MM Settings Tab with Completion Checking Enabled



4.2. Base Address Registers

Table 37. BAR Registers

Parameter	Value	Description
BAR0 Type	**Disabled** **64-bit prefetchable memory** **32-bit non-prefetchable memory**	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 maximum non-prefetchable memory window is 32 bits. 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
BAR1 Size	**256 Bytes – 8 EBytes**	Specifies the size of the address space accessible to the BAR.
BAR2 Size	**256 Bytes – 8 EBytes**	Specifies the size of the address space accessible to the BAR.
BAR3 Size	**256 Bytes – 8 EBytes**	Specifies the size of the address space accessible to the BAR.
BAR4 Size	**256 Bytes – 8 EBytes**	Specifies the size of the address space accessible to the BAR.
BAR5 Size	**256 Bytes – 8 EBytes**	Specifies the size of the address space accessible to the BAR.
BAR1 Type	**Disabled**	

continued...

Parameter	Value	Description
	32-bit non-prefetchable memory	
BAR2 Type	**Disabled** **64-bit prefetchable memory** **32-bit non-prefetchable memory**	
BAR3 Type	**Disabled** **32-bit non-prefetchable memory**	
BAR4 Type	**Disabled** **64-bit prefetchable memory** **32-bit non-prefetchable memory**	
BAR5 Type	**Disabled** **32-bit non-prefetchable memory**	

4.3. Device Identification Registers

The following table lists the default values of the read-only registers in the PCI* Configuration Header Space. You can use the parameter editor to set the values of these registers. At run time, you can change the values of these registers using the optional Hard IP Reconfiguration block signals.

Table 38. PCI Header Configuration Space Registers

Register Name	Default Value	Description
Vendor ID	0x1172	Sets the read-only value of the `Vendor ID` register. This parameter cannot be set to 0xFFFF per the *PCI Express Base Specification*. Address offset: 0x000, bits [15:0].
Device ID	0x0000	Sets the read-only value of the `Device ID` register. Address offset: 0x000, bits [31:16].
Revision ID	0x01	Sets the read-only value of the `Revision ID` register. Address offset: 0x008, bits [7:0].
Class code	0xFF0000	Sets the read-only value of the `Class Code` register. Address offset: 0x008, bits [31:8].
Subsystem Vendor ID	0x0000	Sets the read-only value of `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 value is only used in Root Port variants. Address offset: 0x02C, bits [15:0].
Subsystem Device ID	0x0000	Sets the read-only value of the `Subsystem Device ID` register in the PCI Type 0 Configuration Space. This value is only used in Root Port variants. Address offset: 0x02C, bits [31:16].

4.4. 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.

4.4.1. Device Capabilities

Table 39. Device Registers

Parameter	Possible Values	Default Value	Description
Maximum payload sizes supported	**128 bytes** **256 bytes** **512 bytes**	512 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. A 128-byte read request size results in the lowest latency for typical systems. Address: 0x074.

4.4.2. Link Capabilities

Table 40. 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 Port only. It should not be changed.
Slot clock configuration	**On/Off**	When you turn this option **On**, indicates that the Endpoint 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.

4.4.3. MSI and MSI-X Capabilities

Table 41. MSI and MSI-X Capabilities

Parameter	Value	Description
PF0 Enable MSI	**On/Off**	When **On**, adds the MSI functionality for PF0. Address: 0x050, bits [31:16].
MSI-X Capabilities		
PF0 Enable MSI-X	**On/Off**	When **On**, adds the MSI-X functionality for PF0.
	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. Address offset: 0x068, bits [26:16]
Table offset	[31:0]	Points to the base address 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.

continued...

Parameter	Value	Description
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. (3)
PBA BAR indicator	[2:0]	Specifies which one of a function's Base Address registers, located beginning at 0x10 in Configuration Space, is used to map the MSI-X PBA into memory space. This field is read-only in the MSI-X Capability Structure. Legal range is 0–5.

4.4.4. Power Management

Table 42. Power Management Parameters

Parameter	Value	Description
Endpoint L0s acceptable latency	**Maximum of 64 ns** **Maximum of 128 ns** **Maximum of 256 ns** **Maximum of 512 ns** **Maximum of 1 us** **Maximum of 2 us** **Maximum of 4 us** **No limit**	This design parameter specifies the maximum acceptable latency that the device can tolerate to exit the L0s state for any links between the device and the root complex. It sets the read-only value of the Endpoint L0s acceptable latency field of the `Device Capabilities Register` (0x084). This Endpoint does not support the L0s or L1 states. However, in a switched system there may be links connected to switches that have L0s and L1 enabled. This parameter is set to allow system configuration software to read the acceptable latencies for all devices in the system and the exit latencies for each link to determine which links can enable Active State Power Management (ASPM). This setting is disabled for Root Ports. The default value of this parameter is 64 ns. This is the safest setting for most designs.
Endpoint L1 acceptable latency	**Maximum of 1 us** **Maximum of 2 us** **Maximum of 4 us** **Maximum of 8 us** **Maximum of 16 us** **Maximum of 32 us** **Maximum of 64 us** **No limit**	This value indicates the acceptable latency that an Endpoint can withstand in the transition from the L1 to L0 state. It is an indirect measure of the Endpoint's internal buffering. It sets the read-only value of the Endpoint L1 acceptable latency field of the `Device Capabilities Register`. This Endpoint does not support the L0s or L1 states. However, 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 the safest setting for most designs.

(3) 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.

4.4.5. Vendor Specific Extended Capability (VSEC)

Table 43. VSEC

Parameter	Value	Description
User ID register from the Vendor Specific Extended Capability	Custom value	Sets the read-only value of the 16-bit User ID register from the Vendor Specific Extended Capability. This parameter is only valid for Endpoints.

4.5. Configuration, Debug and Extension Options

Table 44. Configuration, Debug and Extension Options

Parameter	Value	Description
Enable hard IP 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*. With this parameter set to **On**, the hip_reconfig_clk port is visible on the block symbol of the pcie_s10_hip_avmm_bridge_0 component. In the System Contents window, connect a clock source to this hip_reconfig_clk port. For example, you can export hip_reconfig_clk and drive it with a free-running clock on the board whose frequency is in the range of 100 to 125 MHz. Alternatively, if your design includes a clock bridge driven by such a free-running clock (as is the case with the DMA example design), the out_clk of the clock bridge can be used to drive hip_reconfig_clk.
Enable transceiver dynamic reconfiguration	**On/Off**	When **On**, provides an Avalon-MM interface that software can drive to change the values of transceiver registers. With this parameter set to **On**, the xcvr_reconfig_clk, reconfig_pll0_clk, and reconfig_pll1_clk ports are visible on the block symbol of the pcie_s10_hip_avmm_bridge_0 component. In the System Contents window, connect a clock source to these ports. For example, you can export these ports and drive them with a free-running clock on the board whose frequency is in the range of 100 to 125 MHz. Alternatively, if your design includes a clock bridge driven by such a free-running clock (as is the case with the DMA example design), the out_clk of the clock bridge can be used to drive these ports.
Enable Native PHY, ATXPLL, and fPLL ADME for Transceiver Toolkit	**On/Off**	When **On**, the generated IP includes an embedded Altera Debug Master Endpoint (ADME) that connects internally to an Avalon-MM slave interface for dynamic reconfiguration. The ADME can access the transceiver reconfiguration space. It can perform certain test and debug functions via JTAG using the System Console.
Enable PCIe Link Inspector	**On/Off**	When **On**, the PCIe Link Inspector is enabled. Use this interface to monitor the PCIe link at the Physical, Data Link and Transaction layers. You can also use the PCIe Link Inspector to reconfigure some transceiver registers. You must turn on **Enable transceiver dynamic reconfiguration**, **Enable dynamic reconfiguration of PCIe read-only registers** and **Enable Native PHY, ATXPLL, and fPLL ADME for Transceiver Toolkit** to use this feature. For more information about using the PCIe Link Inspector refer to *Link Inspector Hardware* in the *Troubleshooting and Observing Link Status* appendix.

4.6. PHY Characteristics

Table 45. PHY Characteristics

Parameter	Value	Description
VCCR/VCCT supply voltage for the transceiver	**1_1V** **1_0V**	Allows you to select the preferred voltage for the transceivers.

4.7. Example Designs

Table 46. Example Designs

Parameter	Value	Description
Currently Selected Example Design	The generated example design is based on parameterization.	If you enable either/both of the Read Data Mover and Write Data Mover, the generated example design is a **DMA** design, which includes a direct memory access application. This application includes upstream and downstream transactions. Otherwise, if you enable the Bursting Slave option, the generated design example is a Bursting Avalon-MM Slave (BAS) design. If you do not enable either of the options above, the generated design example is a **PIO** design.
Simulation	**On/Off**	When **On**, the generated output includes a simulation model.
Select simulation Root Complex BFM	**Intel FPGA BFM** **Third-party BFM**	The Intel FPGA Root Complex BFM does not support Gen3 x16. Using this BFM makes the design downtrain to Gen3 x8.
Synthesis	**On/Off**	When **On**, the generated output includes a synthesis model.
Generated HDL format	**Verilog/VHDL**	Only Verilog HDL is available in the current release.
Target Development Kit	**None** **Stratix 10 GX H-Tile Production FPGA Development Kit** **Stratix 10 MX H-Tile Production FPGA Development Kit**	Select the appropriate development board. If you select one of the development boards, system generation overwrites the device you selected with the device on that development board. *Note:* If you select **None**, system generation does not make any pin assignments. You must make the assignments in the `.qsf` file.

5. Designing with the IP Core

5.1. Generation

You can use the the Quartus Prime Pro Edition IP Catalog or the Platform Designer to define and generate an Intel L-/H-Tile Avalon-MM+ for PCI Express IP custom component.

5.2. Simulation

The Quartus Prime Pro Edition software optionally generates a functional simulation model, a testbench or design example, and vendor-specific simulator setup scripts when you generate your parameterized PCI Express IP core. For Endpoints, the generation creates a Root Port BFM. There is no support for Root Ports in this release of the Quartus Prime Pro Edition.

The Quartus Prime Pro Edition supports the following simulators.

Table 47. Supported Simulators

Vendor	Simulator	Version	Platform
Aldec	Active-HDL *	10.3	Windows
Aldec	Riviera-PRO *	2016.10	Windows, Linux
Cadence	Incisive Enterprise *	15.20	Linux
Cadence	Xcelium* Parallel Simulator	17.04.014	Linux
Mentor Graphics	ModelSim SE*	10.5c	Windows, Linux
Mentor Graphics	QuestaSim*	10.5c	Windows, Linux
Synopsys	VCS/VCS MX*	2016,06-SP-1	Linux

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

Related Information

- Introduction to Intel FPGA IP Cores, Simulating Intel FPGA IP Cores
- Simulation Quick-Start

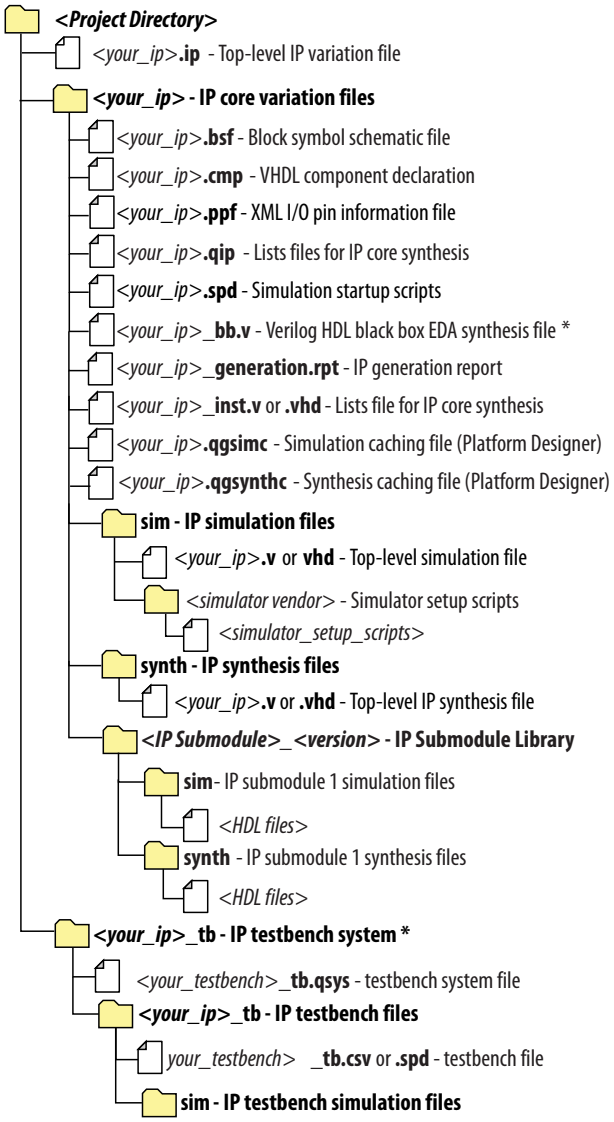
Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

5.3. IP Core Generation Output (Quartus Prime Pro Edition)

The Quartus Prime Pro Edition software generates the following output file structure for individual IP cores that are not part of a Platform Designer system.

Figure 16. Individual IP Core Generation Output (Quartus Prime Pro Edition)



* If supported and enabled for your IP core variation.

Table 48. Output Files of Intel FPGA IP Generation

File Name	Description
<*your_ip*>.ip	Top-level IP variation file that contains the parameterization of an IP core in your project. If the IP variation is part of a Platform Designer system, the parameter editor also generates a .qsys file.
<*your_ip*>.cmp	The VHDL Component Declaration (.cmp) file is a text file that contains local generic and port definitions that you use in VHDL design files.
<*your_ip*>_generation.rpt	IP or Platform Designer generation log file. Displays a summary of the messages during IP generation.
<*your_ip*>.qgsimc (Platform Designer systems only)	Simulation caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.qgsynth (Platform Designer systems only)	Synthesis caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.qip	Contains all information to integrate and compile the IP component.
<*your_ip*>.csv	Contains information about the upgrade status of the IP component.
<*your_ip*>.bsf	A symbol representation of the IP variation for use in Block Diagram Files (.bdf).
<*your_ip*>.spd	Input file that ip-make-simscript requires to generate simulation scripts. The .spd file contains a list of files you generate for simulation, along with information about memories that you initialize.
<*your_ip*>.ppf	The Pin Planner File (.ppf) stores the port and node assignments for IP components you create for use with the Pin Planner.
<*your_ip*>_bb.v	Use the Verilog blackbox (_bb.v) file as an empty module declaration for use as a blackbox.
<*your_ip*>_inst.v or _inst.vhd	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<*your_ip*>.regmap	If the IP contains register information, the Quartus Prime software generates the .regmap file. The .regmap file describes the register map information of master and slave interfaces. This file complements the .sopcinfo file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
<*your_ip*>.svd	Allows HPS System Debug tools to view the register maps of peripherals that connect to HPS within a Platform Designer system. During synthesis, the Quartus Prime software stores the .svd files for slave interface visible to the System Console masters in the .sof file in the debug session. System Console reads this section, which Platform Designer queries for register map information. For system slaves, Platform Designer accesses the registers by name.
<*your_ip*>.v <*your_ip*>.vhd	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
/mentor/	Contains a msim_setup.tcl script to set up and run a ModelSim simulation.
/aldec/	Contains a Riviera*-PRO script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs/ /synopsys/vcsmx/	Contains a shell script vcs_setup.sh to set up and run a VCS* simulation. Contains a shell script vcsmx_setup.sh and synopsys_sim.setup file to set up and run a VCS MX* simulation.

continued...

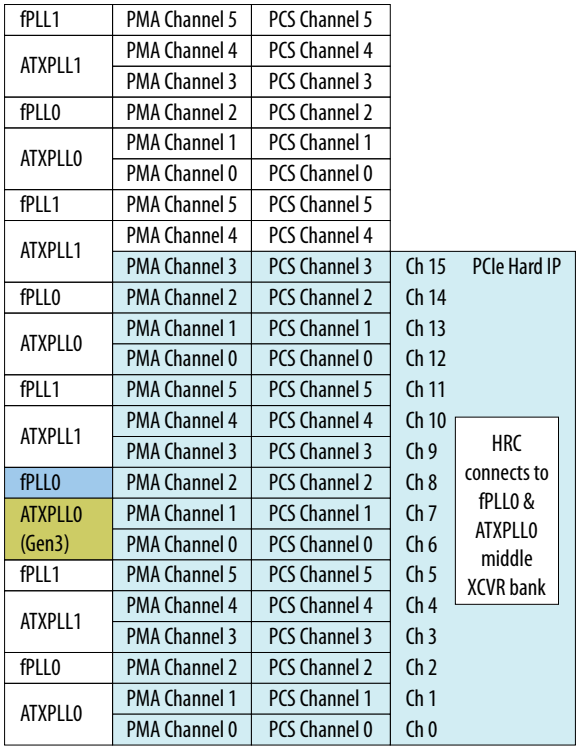
File Name	Description
/cadence/	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSIM simulation.
/submodules/	Contains HDL files for the IP core submodule.
/<IP submodule>/	Platform Designer generates /synth and /sim sub-directories for each IP submodule directory that Platform Designer generates.

5.4. Channel Layout and PLL Usage

The following figure shows the channel layout and connection between the Hard Reset Controller (HRC) and PLLs for the Gen3 x16 variant supported by the Intel L-/H-Tile Avalon-MM+ for PCI Express IP. The channel layout is the same for the Avalon-ST and Avalon-MM interfaces to the Application Layer.

Note: The Hard Reset Controller drives the lower 16 channels. The eight remaining channels are available for other protocols. Refer to *Channel Availability* for more information.

Figure 17. Gen3 x16 Variant



6. Registers

The Intel L-/H-Tile Avalon-MM+ for PCI Express IP does not define any register in addition to the registers defined by the PCIe Hard IP and the PLLs.

6.1. Configuration Space Registers

Table 49. Correspondence between Configuration Space Capability Structures and the PCIe Base Specification Description

Byte Address	Configuration Space Register	Corresponding Section in PCIe Specification
0x000-0x03C	PCI Header Type 0 Configuration Registers	Type 0 Configuration Space Header
0x040-0x04C	Power Management	PCI Power Management Capability Structure
0x050-0x05C	MSI Capability Structure	MSI Capability Structure, see also and *PCI Local Bus Specification*
0x060-0x06C	Reserved	N/A
0x070-0x0A8	PCI Express Capability Structure	PCI Express Capability Structure
0x0B0-0x0B8	MSI-X Capability Structure	MSI-X Capability Structure, see also and *PCI Local Bus Specification*
0x0BC-0x0FC	Reserved	N/A
0x100-0x134	Advanced Error Reporting (AER) (for PFs only)	Advanced Error Reporting Capability
0x138-0x184	Reserved	N/A
0x188-0x1B0	Secondary PCI Express Extended Capability Header	PCI Express Extended Capability
0x1B4	Reserved	N/A
0x1B8-0x1F4	SR-IOV Capability Structure	SR-IOV Extended Capability Header in *Single Root I/O Virtualization and Sharing Specification, Rev, 1.1*
0x1F8-0x1D0	Transaction Processing Hints (TPH) Requester Capability	TLP Processing Hints (TPH)
0x1D4-0x280	Reserved	N/A
0x284-0x288	Address Translation Services (ATS) Capability Structure	Address Translation Services Extended Capability (ATS) in *Single Root I/O Virtualization and Sharing Specification, Rev. 1.1*
0xB80-0xBFC	Intel-Specific	Vendor-Specific Header (Header only)
0xC00	Optional Custom Extensions	N/A
0xC00	Optional Custom Extensions	N/A

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Table 50. Summary of Configuration Space Register Fields

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x000	Device ID, Vendor ID	Type 0 Configuration Space Header
0x004	Status, Command	Type 0 Configuration Space Header
0x008	Class Code, Revision ID	Type 0 Configuration Space Header
0x00C	Header Type, Cache Line Size	Type 0 Configuration Space Header
0x010	Base Address 0	Base Address Registers
0x014	Base Address 1	Base Address Registers
0x018	Base Address 2	Base Address Registers
0x01C	Base Address 3	Base Address Registers
0x020	Base Address 4	Base Address Registers
0x024	Base Address 5	Base Address Registers
0x028	Reserved	N/A
0x02C	Subsystem ID, Subsystem Vendor ID	Type 0 Configuration Space Header
0x030	Reserved	N/A
0x034	Capabilities Pointer	Type 0 Configuration Space Header
0x038	Reserved	N/A
0x03C	Interrupt Pin, Interrupt Line	Type 0 Configuration Space Header
0x040	PME_Support, D1, D2, etc.	PCI Power Management Capability Structure
0x044	PME_en, PME_Status, etc.	Power Management Status and Control Register
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
0x0B0	MSI-X Message Control Next Cap Ptr Capability ID	MSI and MSI-X Capability Structures
0x0B4	MSI-X Table Offset BIR	MSI and MSI-X Capability Structures
0x0B8	Pending Bit Array (PBA) Offset BIR	MSI and MSI-X Capability Structures
0x100	PCI Express Enhanced Capability Header	Advanced Error Reporting Enhanced Capability Header
0x104	Uncorrectable Error Status Register	Uncorrectable Error Status Register
0x108	Uncorrectable Error Mask Register	Uncorrectable Error Mask Register
0x10C	Uncorrectable Error Mask Register	Uncorrectable Error Severity Register
0x110	Correctable Error Status Register	Correctable Error Status Register
0x114	Correctable Error Mask Register	Correctable Error Mask Register
0x118	Advanced Error Capabilities and Control Register	Advanced Error Capabilities and Control Register
0x11C	Header Log Register	Header Log Register
0x12C	Root Error Command	Root Error Command Register

continued...

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x130	Root Error Status	Root Error Status Register
0x134	Error Source Identification Register Correctable Error Source ID Register	Error Source Identification Register
0x188	Next Capability Offset, PCI Express Extended Capability ID	Secondary PCI Express Extended Capability
0x18C	Enable SKP OS, Link Equalization Req, Perform Equalization	Link Control 3 Register
0x190	Lane Error Status Register	Lane Error Status Register
0x194:0x1B0	Lane Equalization Control Register	Lane Equalization Control Register
0xB80	VSEC Capability Header	Vendor-Specific Extended Capability Header
0xB84	VSEC Length, Revision, ID	Vendor-Specific Header
0xB88	Intel Marker	Intel-Specific Registers
0xB8C	JTAG Silicon ID DW0	
0xB90	JTAG Silicon ID DW1	
0xB94	JTAG Silicon ID DW2	
0xB98	JTAG Silicon ID DW3	
0xB9C	User Device and Board Type ID	
0xBA0:0xBAC	Reserved	
0xBB0	General Purpose Control and Status Register	
0xBB4	Uncorrectable Internal Error Status Register	
0xBB8	Uncorrectable Internal Error Mask Register	
0xBBC	Correctable Error Status Register	
0xBC0	Correctable Error Mask Register	
0xBC4:BD8	Reserved	N/A
0xC00	Optional Custom Extensions	N/A

Related Information

- PCI Express Base Specification 3.0
- PCI Local Bus Specification

6.1.1. Register Access Definitions

This document uses the following abbreviations when describing register access.

Table 51. Register Access Abbreviations

Sticky bits are not initialized or modified by hot reset or function-level reset.

Abbreviation	Meaning
RW	Read and write access
RO	Read only
WO	Write only

continued...

Abbreviation	Meaning
RW1C	Read write 1 to clear
RW1CS	Read write 1 to clear sticky
RWS	Read write sticky

6.1.2. PCI Configuration Header Registers

The *Correspondence between Configuration Space Registers and the PCIe Specification* lists the appropriate section of the *PCI Express Base Specification* that describes these registers.

Figure 18. Configuration Space Registers Address Map

End Point Capability Structure	Required/Optional	Starting Byte Offset
PCI Header Type 0	Required	0x00
PCI Power Management	Required	0x40
MSI	Optional	0x50
PCI Express	Required	0x70
MSI-X	Optional	0xB0
AER	Required	0x100
Secondary PCIe	Required	0x188
VSEC	Required	0xB80
Custom Extensions	Optional	0xC00

Figure 19. PCI Configuration Space Registers - Byte Address Offsets and Layout

	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	Reserved			
0x034	Reserved			Capabilities Pointer
0x038	Reserved			
0x03C	0x00		Interrupt Pin	Interrupt Line

Related Information

PCI Express Base Specification 3.0

6.1.3. 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 20. Power Management Capability Structure - Byte Address Offsets and Layout

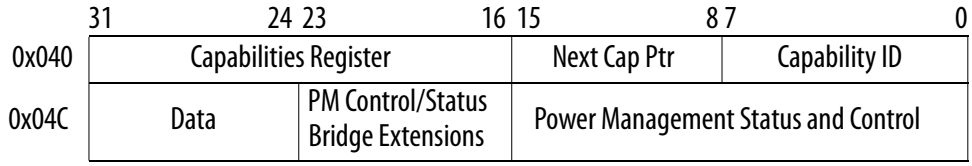


Figure 21. MSI Capability Structure

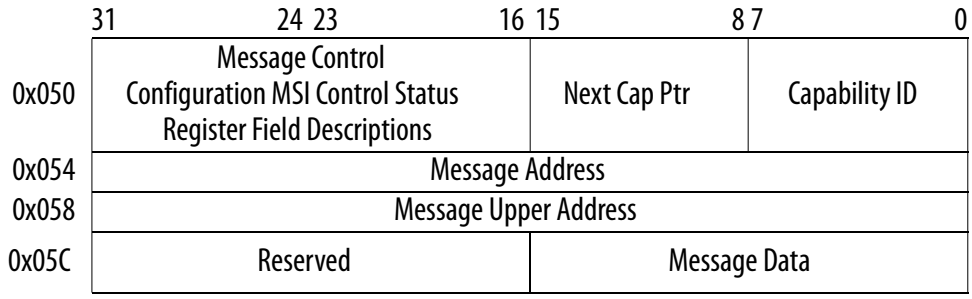


Figure 22. PCI Express Capability Structure - Byte Address Offsets and Layout

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

	31 24	23 16	15 8	7 0
0x070	PCI Express Capabilities Register		Next Cap Pointer	PCI Express Capabilities ID
0x074	Device Capabilities			
0x078	Device Status		Device Control	
0x07C	Link Capabilities			
0x080	Link Status		Link Control	
0x084	Slot Capabilities			
0x088	Slot Status		Slot Control	
0x08C	Root Capabilities		Root Control	
0x090	Root Status			
0x094	Device Compatibilities 2			
0x098	Device Status 2		Device Control 2	
0x09C	Link Capabilities 2			
0x0A0	Link Status 2		Link Control 2	
0x0A4	Slot Capabilities 2			
0x0A8	Slot Status 2		Slot Control 2	

Figure 23. MSI-X Capability Structure

	31 24	23 16	15 8	7 3	2 0
0x0B0	Message Control		Next Cap Ptr	Capability ID	
0x0B4	MSI-X Table Offset				MSI-X Table BAR Indicator
0x0B8	MSI-X Pending Bit Array (PBA) Offset				MSI-X Pending Bit Array - BAR Indicator

Figure 24. PCI Express AER Extended Capability Structure

	31	16 15	0
0x100	PCI Express Enhanced Capability Register		
0x104	Uncorrectable Error Status Register		
0x108	Uncorrectable Error Mask Register		
0x10C	Uncorrectable Error Severity Register		
0x110	Correctable Error Status Register		
0x114	Correctable Error Mask Register		
0x118	Advanced Error Capabilities and Control Register		
0x11C	Header Log Register		
0x12C	Root Error Command Register		
0x130	Root Error Status Register		
0x134	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.

Related Information

- PCI Express Base Specification 3.0
- PCI Local Bus Specification

6.1.4. Intel Defined VSEC Capability Header

The figure below shows the address map and layout of the Intel defined VSEC Capability.

Figure 25. Vendor-Specific Extended Capability Address Map and Register Layout

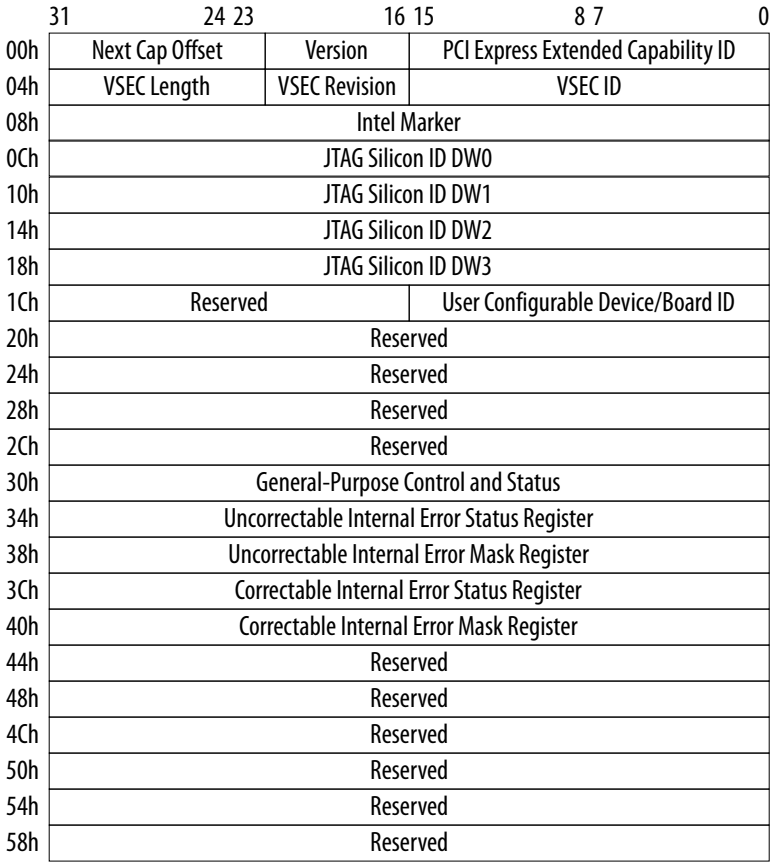


Table 52. Intel-Defined VSEC Capability Header - 0xB80

Bits	Register Description	Default Value	Access
[31:20]	Next Capability Pointer: Value is the starting address of the next Capability Structure implemented. Otherwise, NULL.	Variable	RO
[19:16]	Version. PCIe specification defined value for VSEC version.	1	RO
[15:0]	PCI Express Extended Capability ID. PCIe specification defined value for VSEC Capability ID.	0x000B	RO

6.1.4.1. Intel Defined Vendor Specific Header

Table 53. Intel defined Vendor Specific Header - 0xB84

Bits	Register Description	Default Value	Access
[31:20]	VSEC Length. Total length of this structure in bytes.	0x5C	RO
[19:16]	VSEC. User configurable VSEC revision.	Not available	RO
[15:0]	VSEC ID. User configurable VSEC ID. You should change this ID to your Vendor ID.	0x1172	RO

6.1.4.2. Intel Marker

Table 54. Intel Marker - 0xB88

Bits	Register Description	Default Value	Access
[31:0]	Intel Marker - An additional marker for standard Intel programming software to be able to verify that this is the right structure.	0x41721172	RO

6.1.4.3. JTAG Silicon ID

This read only register returns the JTAG Silicon ID. The Intel Programming software uses this JTAG ID to make ensure that it is programming the SRAM Object File (`*.sof`) .

Table 55. JTAG Silicon ID - 0xB8C-0xB98

Bits	Register Description	Default Value (4)	Access
[31:0]	JTAG Silicon ID DW3	Unique ID	RO
[31:0]	JTAG Silicon ID DW2	Unique ID	RO
[31:0]	JTAG Silicon ID DW1	Unique ID	RO
[31:0]	JTAG Silicon ID DW0	Unique ID	RO

6.1.4.4. User Configurable Device and Board ID

Table 56. User Configurable Device and Board ID - 0xB9C

Bits	Register Description	Default Value	Access
[15:0]	Allows you to specify ID of the **.sof** file to be loaded.	From configuration bits	RO

(4) Because the Silicon ID is a unique value, it does not have a global default value.

6.1.5. Uncorrectable Internal Error Status Register

This register reports the status of the internally checked errors that are uncorrectable.When these specific errors are enabled by the `Uncorrectable Internal Error Mask` register, they are forwarded as Uncorrectable Internal Errors. This register is for debug only. Only use this register to observe behavior, not to drive logic custom logic.

Table 57. Uncorrectable Internal Error Status Register - 0xBB4

This register is for debug only. It should only be used to observe behavior, not to drive custom logic.

Bits	Register Description	Reset Value	Access
[31:13]	Reserved.	0	RO
[12]	Debug bus interface (DBI) access error status.	0	RW1CS
[11]	ECC error from Config RAM block.	0	RW1CS
[10]	Uncorrectable ECC error status for Retry Buffer.	0	RO
[9]	Uncorrectable ECC error status for Retry Start of the TLP RAM.	0	RW1CS
[8]	RX Transaction Layer parity error reported by the IP core.	0	RW1CS
[7]	TX Transaction Layer parity error reported by the IP core.	0	RW1CS
[6]	Internal error reported by the FPGA.	0	RW1CS
[5:4]	Reserved.	0	RW1CS
[3]	Uncorrectable ECC error status for RX Buffer Header #2 RAM.	0	RW1CS
[2]	Uncorrectable ECC error status for RX Buffer Header #1 RAM.	0	RW1CS
[1]	Uncorrectable ECC error status for RX Buffer Data RAM #2.	0	RW1CS
[0]	Uncorrectable ECC error status for RX Buffer Data RAM #1.	0	RW1CS

6.1.6. Uncorrectable Internal Error Mask Register

The `Uncorrectable Internal Error Mask` register controls which errors are forwarded as internal uncorrectable errors.

Table 58. Uncorrectable Internal Error Mask Register - 0xBB8

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
[31:13]	Reserved.	1b'0	RO
[12]	Mask for Debug Bus Interface.	1b'1	RO
[11]	Mask for ECC error from Config RAM block.	1b'1	RWS
[10]	Mask for Uncorrectable ECC error status for Retry Buffer.	1b'1	RO
[9]	Mask for Uncorrectable ECC error status for Retry Start of TLP RAM.	1b'1	RWS
[8]	Mask for RX Transaction Layer parity error reported by IP core.	1b'1	RWS
[7]	Mask for TX Transaction Layer parity error reported by IP core.	1b'1	RWS
[6]	Mask for Uncorrectable Internal error reported by the FPGA.	1b'1	RO

continued...

Bits	Register Description	Reset Value	Access
[5]	Reserved.	1b'0	RWS
[4]	Reserved.	1b'1	RWS
[3]	Mask for Uncorrectable ECC error status for RX Buffer Header #2 RAM.	1b'1	RWS
[2]	Mask for Uncorrectable ECC error status for RX Buffer Header #1 RAM.	1b'1	RWS
[1]	Mask for Uncorrectable ECC error status for RX Buffer Data RAM #2.	1b'1	RWS
[0]	Mask for Uncorrectable ECC error status for RX Buffer Data RAM #1.	1b'1	RWS

6.1.7. Correctable Internal Error Status Register

Table 59. Correctable Internal Error Status Register - 0xBBC

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. 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
[31:12]	Reserved.	0	RO
[11]	Correctable ECC error status for Config RAM.	0	RW1CS
[10]	Correctable ECC error status for Retry Buffer.	0	RW1CS
[9]	Correctable ECC error status for Retry Start of TLP RAM.	0	RW1CS
[8]	Reserved.	0	RO
[7]	Reserved.	0	RO
[6]	Internal Error reported by FPGA.	0	RW1CS
[5]	Reserved	0	RO
[4]	PHY Gen3 SKP Error occurred. Gen3 data pattern contains SKP pattern (8'b10101010) is misinterpreted as a SKP OS and causing erroneous block realignment in the PHY.	0	RW1CS
[3]	Correctable ECC error status for RX Buffer Header RAM #2.	0	RW1CS
[2]	Correctable ECC error status for RX Buffer Header RAM #1.	0	RW1CS
[1]	Correctable ECC error status for RX Buffer Data RAM #2.	0	RW1CS
[0]	Correctable ECC error status for RX Buffer Data RAM #1.	0	RW1CS

6.1.8. Correctable Internal Error Mask Register

Table 60. Correctable Internal Error Status Register - 0xBBC

The `Correctable Internal Error Status` register controls which errors are forwarded as Internal Correctable Errors.

Bits	Register Description	Reset Value	Access
[31:12]	Reserved.	0	RO
[11]	Mask for correctable ECC error status for Config RAM.	0	RWS
[10]	Mask for correctable ECC error status for Retry Buffer.	1	RWS

continued...

Bits	Register Description	Reset Value	Access
[9]	Mask for correctable ECC error status for Retry Start of TLP RAM.	1	RWS
[8]	Reserved.	0	RO
[7]	Reserved.	0	RO
[6]	Mask for internal Error reported by FPGA.	0	RWS
[5]	Reserved	0	RO
[4]	Mask for PHY Gen3 SKP Error.	1	RWS
[3]	Mask for correctable ECC error status for RX Buffer Header RAM #2.	1	RWS
[2]	Mask for correctable ECC error status for RX Buffer Header RAM #1.	1	RWS
[1]	Mask for correctable ECC error status for RX Buffer Data RAM #.	1	RWS
[0]	Mask for correctable ECC error status for RX Buffer Data RAM #1.	1	RWS

7. Design Example and Testbench

7.1. Overview of the Example Design

The available example design is for an Endpoint, with a single function and no SR-IOV support.

This DMA example design includes a DMA Controller and an on-chip memory to exercise the Data Movers, and a Traffic Generator and Checker to exercise the Bursting Slave.

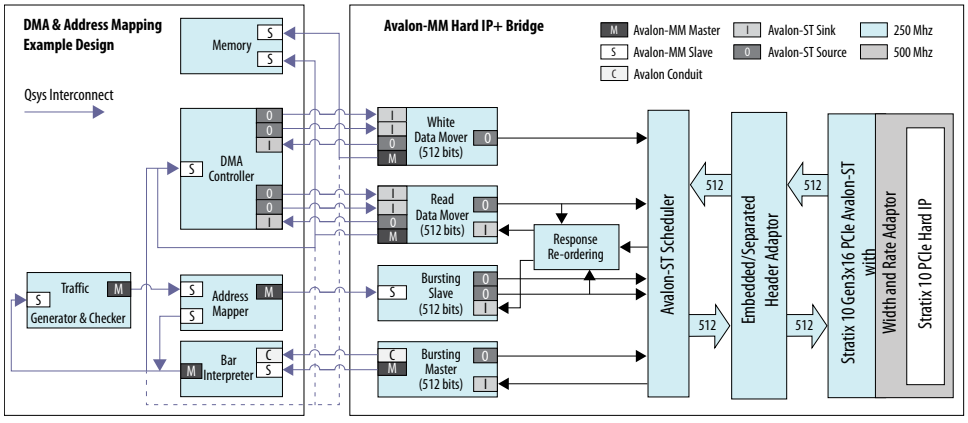
The example design also connects the Bursting Master to the on-chip memory to allow high-throughput transfers should the host or some other component of the PCIe system be capable of initiating such transfers (e.g. a Root Complex with a DMA engine).

An address translation module is inserted between the Traffic Generator and the Bursting Slave to show an example of how address translation can be performed between the Avalon and PCIe address spaces.

The on-chip memory that the Data Movers and the Bursting Master connect to is a dual-port memory to allow full-duplex data movement.

The Bursting Master connects to a BAR Interpreter module, which combines the address and BAR number and allows the Bursting Master to control the DMA Controller, the Traffic Generator and the Address Mapper. The BAR Interpreter also connects the Bursting Master to the dual-port memory.

Figure 26. DMA Example Design



Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

The example design is generated dynamically based on the selected variation of the Intel L-/H-Tile Avalon-MM+ for PCI Express IP. However, some of the user's parameter selections may need to be overwritten to ensure proper functionality. A warning appears when such a need arises.

The DMA Controller is instantiated for variations that implement at least one of the Data Movers and the Bursting Master. The Traffic Generator and Checker module is instantiated for variations that implement at least the Bursting Slave and the Bursting Master.

You cannot generate an example design for variations that do not implement the Bursting Master as it is necessary for all flavors of the example design. A warning appears to explain that the Bursting Master with appropriate BAR settings must be added in order to generate the example design.

There are three variations currently supported:

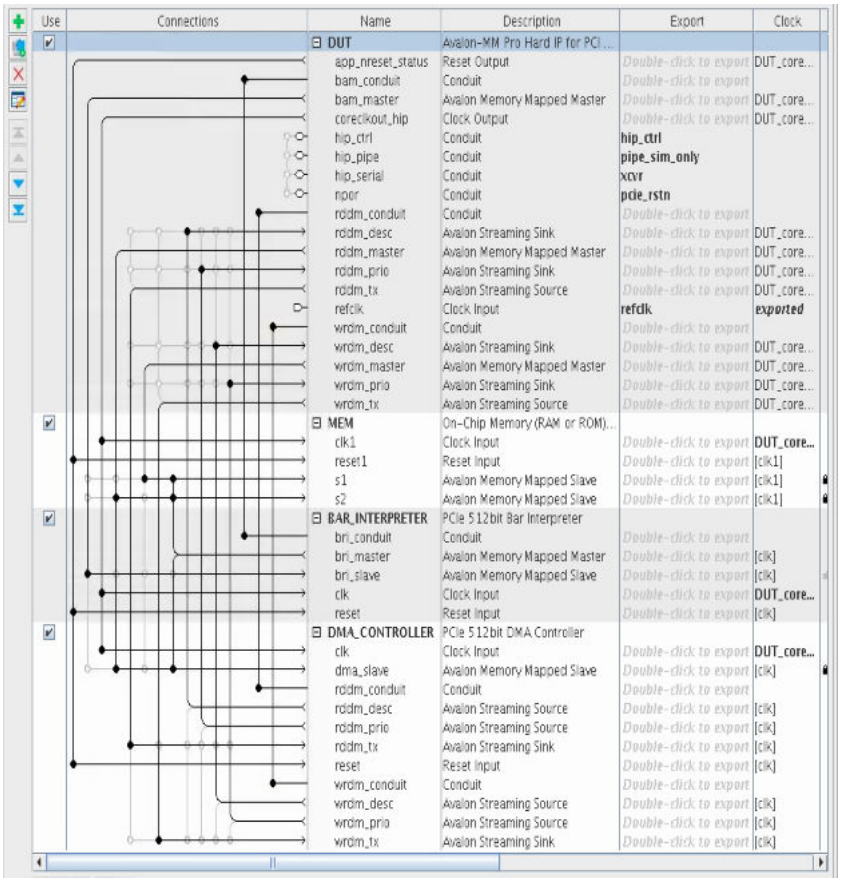
- **PIO**: This is the variation where only the Bursting Master is enabled. The design example simulation exercises the Bursting Master to perform simple one dword Reads and Writes to the on-chip memory. The BAR Interpreter and on-chip memory are also included in this variation.
- **DMA**: This is the variation where the Bursting Master, and both of the Read Data Mover and Write Data Mover are enabled. Software sends instructions via the Bursting Master to the Read or Write Data Movers to initiate DMA Reads or Writes to the system memory. The BAR Interpreter, on-chip memory and DMA Controller are also included.
- **BAS**: This is the variation where the Bursting Master and Bursting Slave are enabled.
  - — Software writes an instruction via the Bursting Master to the Traffic Generator to generate a block of data, which is then transmitted to the Bursting Slave. Upon receiving the data, the Bursting Slave performs a Memory Write to transfer the data to the PCIe system memory.
  - — Software then writes another instruction via the Bursting Master to the Traffic Checker, which issues a Read to the Bursting Slave. The Bursting Slave then forms a Memory Read request to fetch the data from the PCIe system memory.

  The BAR Interpreter, on-chip memory, Traffic Generator and Address Mapper are included in this variation.

Note: Beginning with the 17.1 release, the Quartus Prime Pro Edition software dynamically generates example designs for the parameters you specify in the parameter editor. Consequently, the Quartus Prime Pro Edition installation directory no longer provides static example designs for Stratix 10 devices. Static example designs are available for earlier device families, including Arria® 10 and Cyclone® 10 devices.

The following figure shows the system-level view in Platform Designer of the DMA example design.

Figure 27. Stratix 10 Intel L-/H-Tile Avalon-MM+ for PCI Express IP DMA Example Design in Platform Designer



7.1.1. Block Descriptions

7.1.1.1. DMA Controller (DMA Example Design Only)

The DMA Controller in this example design consists of six addressable queues: two write-only queues and one read-only queue each for the Read Data Mover and the Write Data Mover. In addition, the DMA Controller has two MSI control registers for each Data Mover module.

The write-only queues directly feed into the Data Movers' normal and priority descriptor queues. The read-only queues read directly from the Data Movers' status queues.

The MSI control registers control whether MSI generation is enabled and defines the address and data to be used for the MSI.

The entire example design is in the `coreclkout_hip` clock domain.

Note: The Intel L-/H-Tile Avalon-MM+ for PCI Express IP core does not include an internal DMA Controller. You can use the DMA Controller included in the example design that you can generate, or provide your own DMA Controller.

7.1.1.1.1. Register Set

The registers in the DMA Controller are 512-bit wide to match the data path width of the Bursting Master's and Read Data Mover's Avalon-MM Master. This allows the Read Data Mover to write a descriptor in a single cycle if desired.

Table 61. Register Set of the DMA Controller

Offset	Name	Access	Description
0x000	WDN	R/W	write: descriptor for the Write Data Mover normal descriptor queue read: readiness and fill level of the Write Data Mover normal descriptor queue
0x200	WDP	R/W	write: descriptor for the Write Data Mover priority descriptor queue read: readiness and fill level of the Write Data Mover priority descriptor queue
0x400	WS	RO	Write Data Mover status queue
0x600	WI	R/W	Write Data Mover interrupt control register
0x800	RDN	R/W	write: descriptor for the Read Data Mover normal descriptor queue read: readiness and fill level of the Read Data Mover normal descriptor queue
0xA00	RDP	R/W	write: descriptor for the Read Data Mover priority descriptor queue read: readiness and fill level of the Read Data Mover priority descriptor queue
0xC00	RS	RO	Read Data Mover status queue
0xE00	RI	R/W	Read Data Mover interrupt control register

For the data written to the descriptor queue registers, use the same format and content as the data on the corresponding Avalon-ST interfaces of the Data Movers. The least significant of the application specific bits indicates whether an interrupt should be issued when processing of that descriptor completes.

The DMA Controller double buffers the write-only queues so that the descriptors can be built one DWORD at a time if required, for example by a 32-bit host controller. The content of the register is transferred to the Data Movers' Avalon-ST input when the most significant DWORD is written.

Note: If you write to the DMA Controller's register file with a write burst, all the data (up to 512 bytes) goes to the register addressed in the first cycle of the burst. This behavior enables writing multiple descriptors to one of the queues with a single burst write transaction. The downside is that you cannot write to adjacent registers with a single burst.

Attempting to write to a descriptor queue when the corresponding Data Mover's `ready` signal is not asserted causes the DMA Controller to assert its `waitrequest` signal until `ready` is asserted. You must make sure the Read Data Mover does not attempt to write to the same queue that it is processing while the queue is full, as that would lead to a deadlock. For more details on deadlocks, refer to the section *Deadlock Risk and Avoidance*.

You can find the status of the `ready` signal of a descriptor queue interface by checking the `ready` bit (bit [31]) of the queue registers. In addition, bits [7:0] of the queue registers indicate the approximate fill level of the queues. The other bits of the queue registers are set to 0.

Only the least significant DWORD of the WS and RS registers contains significant information. The other bits are set to 0.

The format and content of the status queues are identical to the corresponding Avalon-ST interfaces of the Data Movers with the addition of bit 31 indicating that the queue is empty. Reading from one of the status queues when it is empty returns 512'h8000_0000.

The format of the WI and RI interrupt control registers is as follows: `{enable, priority, reserved[414:0], msi_msg_data[15:0], reserved[15:0], msi_address[63:0]}`.

The `enable` bit controls whether or not an MSI is sent. The `priority` bit specifies whether to use the priority queue to send the MSI. The MSI memory write TLP also uses the contents of the `msi_msg_data` and `msi_address` fields.

7.1.1.1.2. Deadlock Risk and Avoidance

Under certain circumstances, it is possible for the DMA engine in the example design hardware to get into a deadlock. This section describes the conditions that may lead to a deadlock, and how to avoid them.

When you program the DMA Controller to use the Read Data Mover to fetch too many descriptors for the Read Data Mover descriptor queue, the following loop of back-pressure that leads to a deadlock can occur.

Once the Read Data Mover has transferred enough descriptors through the DMA Controller to its own descriptor queue to fill up the queue, it deasserts its `ready` output. The DMA Controller in turn asserts its `waitrequest` output, thus preventing the Read Data Mover from writing any remaining descriptor to its own queue. After this situation occurs, the Read Data Mover continues to issue MRd read requests, but because the completions can no longer be written to the DMA Controller, the tags associated with these MRd TLPs are not released. The Read Data Mover eventually runs out of tags and stops, having gotten into a deadlock situation.

To avoid this deadlock situation, you can limit the number of descriptors that are fetched at a time. Doing so ensures that the Read Data Mover's descriptor queue never fills up when it is trying to write to its own descriptor queue.

7.1.1.1.3. Interrupts

Two application specific bits (bits [13:12]) of the status words from the Write Data Mover and Read Data Mover Status Avalon-ST Source interfaces control when interrupts are generated.

Table 62. Interrupts Control

Bit [13]	Bit [12]	Action
1	1	Interrupt always
1	0	Interrupt if error
0	1	No interrupt
0	0	No interrupt and drop status word (i.e, do not even write it to the WS or RS status queues)

The DMA Controller makes the decision whether to drop the status word and whether to generate an interrupt as soon as it receives the status word from the Data Mover. When generation of an interrupt is requested, and the corresponding RI or WI register does enable interrupts, the DMA Controller generates the interrupt. It does so by queuing an immediate write to the Write Data Mover's descriptor queue specified in the corresponding interrupt control register using the MSI address and message data provided in that register.You need to make sure that space is always available in the targeted Write Data Mover descriptor queue at any time when an interrupt may get generated. You can do so most easily by using the priority queue only for MSIs.

Setting the interrupt control bits in the immediate write descriptors that the DMA Controller creates to generate MSI interrupts to "No interrupt and drop status word" can avoid an infinite loop of interrupts.

7.1.1.1.4. Using the DMA Controller

To initiate a single DMA transfer, you only need to write a well-formed descriptor to one of the DMA Controller's descriptor queues (WDN, WDP, RDN or RDP).

To initiate a series of DMA transfers, you can prepare a table of descriptors padded to 512 bits each in a memory location accessible to the Read Data Mover. You can then write a single descriptor to the DMA Controller's priority descriptor queue (RDP) register to initiate the DMA transfers. These transfers move the descriptors from the source location in PCIe memory to the desired descriptor queue register.

To transmit an MSI interrupt upon completion of the processing of a descriptor, you must program the DMA Controller's WI or RI register with the desired MSI address and message before writing the descriptor.

7.1.1.2. Traffic Generator and Checker (BAS Example Design Only)

This module creates read and write transactions to exercise the Bursting Slave module. You program it by writing to its control registers through its Control and Status Avalon-MM slave interface.

The traffic that it generates and expects is a sequence of incrementing dwords.

In the checking process, the first dword is accepted as-is, and the following dwords are checked to be incrementing.

7.1.1.2.1. Register Set

The registers in the Traffic Generator are 32-bit wide.

Table 63. Register Set

Offset	Name	Description
0x00	RAdd	Read start address
0x04	RCnt	Read count
0x08	RErr	Read error count
0x0C	RCtl	Read control
0x10	WAdd	Write start address
0x14	WCnt	Write count
0x18	WErr	Reserved (Write error detection not available yet)
0x1C	WCtl	Write control

The RAdd and WAdd registers contain the base addresses that the Traffic Generator reads from and writes to respectively. These are byte addresses and must be Avalon-MM word aligned (i.e. bits [5:0] are assumed to be set to 0).

Write to the RCnt and WCnt registers to specify the number of transfers to execute. Writing a 0 to these registers means non-stop transfers. Reading from one of these registers returns the number of transfers that have occurred since it was last read.

Reading the RErr register returns the number of errors detected since the register was last read. A maximum of one error is counted per clock cycle. Because the write error detection feature is not available yet, you cannot get a valid number of errors by reading the WErr register.

The RCtl and WCtl registers contain fields that define various aspects of the transfers, and start and stop the transfers.

Table 64. RCtl and WCtl Register Bits Descriptions

Bits	Name	Description
[3:0]	target_size	Size of the area of memory to read or write 0: 1 KB 1: 2 KB 2: 4 KB 3: 8 KB 4: 16 KB 5: 32 KB 6: 64 KB 7: 128 KB 8: 256 KB 9: 512 KB 10: 1 MB 11 - 15: Reserved
[7:4]	transfer_size	Size of transfers 0: 1 byte 1: 2 bytes 2: 1 dword (4 bytes) 3: 2 dwords (8 bytes) 4: 4 dwords (16 bytes) 5: 8 dwords (32 bytes) 6: 16 dwords (1 cycle - 64 bytes)

continued...

Bits	Name	Description
		7: 32 dwords (2 cycles - 128 bytes) 8: 48 dwords (3 cycles - 192 bytes) 9: 64 dwords (4 cycles - 256 bytes) 10: 80 dwords (5 cycles - 320 bytes) 11: 96 dwords (6 cycles - 384 bytes) 12: 112 dwords (7 cycles - 448 bytes) 13: 128 dwords (8 cycles - 512 bytes) 14: 16 dwords (2 cycles - 64 bytes). Start at 32 bytes offset from the specified address. 15: Reserved
[30:8]	reserved	Reserved
31	enable	0: stop 1: start

All addresses are read or written sequentially. For example, if the transfer size is one byte and the first target byte address is N, the second transfer targets byte address N +1 (using the appropriate byte enable). If the transfer size is 48 dwords and the first target byte address is N, the second transfer targets byte address N + 192.

If the number of transfers multiplied by the transfer size is larger than the target area (or if the transfers are non-stop), the transfers loop back to the beginning of the target area when they reach its end. For transfer sizes that are not a power of two, the exact behavior at wraparound is not specified. The Traffic Checker detects one error every time the transfers loop back to the beginning of the target area.

The `target_size` parameter should not exceed the size of one address mapping window, or at least should not exceed the size of the address mapping table.

7.1.1.3. Avalon-MM Address to PCIe Address Mapping

The Bursting Slave module transforms read and write transactions on its Avalon-MM interface into PCIe memory read (MRd) and memory write (MWr) request packets. The Bursting Slave uses the Avalon-MM address provided on its 64-bit wide address bus directly as the PCIe address in the TLPs that it creates.

The Bursting Slave, with its 64-bit address bus, uses up the whole Avalon-MM address space and prevents other slaves from being connected to the same bus. In many cases, the user application only needs to access a few relatively small regions of the PCIe address space, and would prefer to dedicate a smaller address space to the Bursting Slave to be able to connect to other slaves.

The example design includes an address mapping module that maps sixteen 1 MB regions of the Avalon-MM memory space into sixteen 1 MB regions of the PCIe address space. The module occupies only 16MB of the Avalon-MM address space, and only needs a 24-bit wide address bus, leaving space for other Avalon-MM slaves.

7.1.1.4. BAR Interpreter

The Bursting Master module transforms PCIe memory read and write request packets received from the PCIe system into Avalon-MM read and write transactions. The offset from the matching BAR is provided as the Avalon-MM address, and the number of the matching BAR is provided in a conduit synchronously with the address.

Although these signals are in a conduit separate from the Avalon-MM master interface, they are synchronous to it and can be treated as extensions of the address bus.

The BAR Interpreter simply concatenates the BAR number to the address bus to form a wider address bus that Platform Designer can now treat as a normal address bus and route to the various slaves connected to the BAR Interpreter.

7.1.2. Programming Model for the Example Design

The programming model for the DMA example design performs the following steps:

1. In system memory, prepare a contiguous set of descriptors. The last of these descriptors is an immediate write descriptor, with the destination address set to some special system memory status location. The descriptor table must start on a 64-byte aligned address. Even though each descriptor is only about 160-bit long, 512 bits are reserved for each descriptor. The descriptors are LSB-aligned in that 512-bit field.

Figure 28. Sample Descriptor Table

	Bits		Offset
	Bits [31:1]	Bit 0	
Read Descriptor 0 Status	Reserved	Done	0x000
Read Descriptor 1 Status	Reserved	Done	0x004
⋮	⋮	⋮	⋮
Write Descriptor 0 Status	Reserved	Done	0x200
Write Descriptor 1 Status	Reserved	Done	0x204
⋮	⋮	⋮	⋮
Read Descriptor 0: Start	SRC_ADDR_LOW		0x400
	SRC_ADDR_HIGH		0x404
	DEST_ADDR_LOW		0x408
	DEST_ADDR_HIGH		0x40C
	DMA_LENGTH		0x410
Read Descriptor 0: End	Reserved		0x414-0x41C
⋮	⋮	⋮	⋮
Read Descriptor 127: Start	SRC_ADDR_LOW		0x13F8
Read Descriptor 127: End	Reserved		0x13FC
Write Descriptor 0: Start	SRC_ADDR_LOW		0x1400
	SRC_ADDR_HIGH		0x1404
	DEST_ADDR_LOW		0x1408
	DEST_ADDR_HIGH		0x140C
	DMA_LENGTH		0x1410
Write Descriptor 0: End	Reserved		0x141C
⋮	⋮	⋮	⋮
Write Descriptor 127: Start	SRC_ADDR_LOW		0x23F8
Write Descriptor 127: End	Reserved		0x23FC

2. In system memory, prepare one more descriptor which reads from the beginning of the descriptors from Step 1 and writes them to a special FIFO Avalon-MM address in FPGA.
3. Write the descriptor in Step 2 to the same special FIFO Avalon-MM address by:

a. Writing one dword at a time, ending with the most significant dword.

b. Writing three dwords of padding and the entire descriptor for a total of eight dwords (the descriptor takes up only five dwords, but CPUs do not typically support single-TLP, five-dword writes).

4. Poll the special status location in system memory to see if the final immediate write has occurred, indicating the DMA completion.

7.1.3. DMA Operations Using the Example Design

7.1.3.1. Read DMA Example

A Read DMA transfers data from the PCIe address space (system memory) to the Avalon-MM address space. It sends Memory Read TLPs upstream, and writes the completion data to local memory in the Avalon-MM address space using the Read Data Mover's Avalon-MM write master interface.

The sequence of steps the example design follows to do a Read DMA is:

1. Prepare a table of descriptors (padded to 512-bit each) to perform the Read operation and put the table into the system memory.
2. Using the BAM, send one descriptor from software containing the address of the descriptor table to the DMA Controller, which forwards it to the Read Data Mover.
3. The Read Data Mover fetches the descriptor table and puts it in a FIFO inside the DMA Controller.
4. The DMA Controller outputs these descriptors to the Read Data Mover based on the readiness of the Read Data Mover (indicated by an asserted rddm_desc_ready_o or rddm_prio_ready_o signal).
5. The Read Data Mover processes the descriptors by fetching data from the system memory, and writing it to the appropriate Avalon-MM memory.
6. The last descriptor processed by the Read Data Mover points to an immediate write descriptor (i.e, a descriptor where the data to be written is inside the descriptor itself) in the system memory. This descriptor's destination address is the Avalon memory address of the DMA Controller's Write Data Mover port. The Read Data Mover fetches this descriptor from system memory and transfers it to the DMA Controller's Write Data Mover Avalon address.
7. The Write Data Mover uses the descriptor from Step 6 to perform an immediate write to the system memory indicating the completion of the Read Data Mover’s data processing.

7.1.3.2. Write DMA Example

A Write DMA transfers data from the Avalon-MM address space to the PCIe address space (system memory). It uses the Write Data Mover's Avalon-MM read master to read data from the Avalon-MM address space and sends it upstream using Memory Write TLPs.

The sequence of steps the example design follows to do a Write DMA is:

1. Prepare a table of descriptors (padded to 512-bit each) to perform the Write operation and put the table into the system memory.
2. Using the BAM, send one descriptor from software containing the address of the descriptor table to the DMA Controller, which forwards it to the Read Data Mover.

3. The Read Data Mover fetches the descriptor table and puts it in a FIFO inside the DMA Controller.
4. The DMA Controller outputs these descriptors to the Write Data Mover based on the readiness of the Write Data Mover (indicated by an asserted wrdm_desc_ready_o or wrdm_prio_ready_o signal).
5. The Write Data Mover processes the descriptors by fetching data from the Avalon-MM memory, and writing it to the appropriate system memory.
6. The Write Data Mover uses the last descriptor in the descriptor table to indicate the completion of the Write Data Mover's data processing. This descriptor is an Immediate Write (the data is inside the descriptor itself) to the system memory indicating the Write Data Mover's operations are done.

7.1.3.3. Using the Traffic Generator (BAS Example Design Only)

To use the Traffic Generator, the host software must follow the following steps:

1. Allocate a block of memory in PCIe space.
2. Program one of the windows in the Address Mapper to point to the block of memory allocated in step 1.
3. Set the Traffic Generator's Write Address register to point to the base of window selected in step 2.
4. Set the Traffic Generator's Write Count to the size of the block of memory allocated in step 1 (or 1MB if the block of memory is larger than 1MB).
5. Write to the Traffic Generator's Write Control register to:
   a. Set the target_size to the size of the block of memory.
   b. Set the transfer size to the desired size, e.g. 512 bytes.
   c. Set the enable bit to start traffic generation.
6. Read the Traffic Generator's Write Count register to check that the desired number of transfers have occurred.

7.1.3.4. Using the Traffic Checker (BAS Example Design Only)

To use the Traffic Checker, the host software must follow the following steps:

1. Allocate a block of memory in PCIe space and initialize the memory with sequential dwords.
2. Program one of the windows in the Address Mapper to point to the block of memory allocated in step 1.

   Steps 1 and 2 can be done by running the Traffic Generator as described in the topic "Using the Traffic Generator".
3. Set the Traffic Checker's Read Address register to point to the base of window selected in step 2.
4. Set the Traffic Checker's Read Count to the size of the block of memory allocated and initialized in step 1 (or 1MB if that area is larger than 1MB).
5. Write to the Traffic Checker's Read Control register to:
   a. Set the target_size to the size of the block of memory.
   b. Set the transfer size to the desired size, e.g. 512 bytes.

c. Set the write_enable bit to start traffic generation.

6. Read the Traffic Checker's Read Count register to check that the desired number of transfers have occurred.
7. Read the Traffic Checker's Error register to check that no error occurred.

7.2. Overview of the Testbench

This testbench simulates up to x16 variants. However, the provided BFM only supports x1 to x8 links. It supports x16 variants by downtraining to x8. To simulate all lanes of a x16 variant, you can create a simulation model to use in a testbench along with a third party BFM such as the Avery BFM. For more information refer to *AN-811: Using the Avery BFM for PCI Express Gen3x16 Simulation on Stratix 10 Devices*.

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

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

This testbench simulates a single Endpoint DUT.

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

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

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

- It is unable to generate or receive Vendor Defined Messages. Some systems generate Vendor Defined Messages and the Application Layer must be designed to process them. The Hard IP block passes these messages on to the Application Layer, which in most cases should ignore them. The Avalon-MM bridge itself ignores Vendor Defined Messages.
- It can only handle received read requests that are less than or equal to the currently set Maximum Payload Size. You can set this parameter in the parameter editor by going to **IP Settings ➤ PCI Express/PCI Capabilities ➤ Device ➤ Maximum payload size**. Many systems are capable of handling larger read requests that are then returned in multiple completions.
- It always returns a single completion for every read request. Some systems split completions on every 64-byte address boundary.
- It always returns completions in the same order the read requests were issued. Some systems generate the completions out-of-order.
- It is unable to generate zero-length read requests that some systems generate as flush requests following some write transactions. The Application Layer must be capable of generating the completions to the zero length read requests.
- It uses a fixed credit allocation.
- It does not support multi-function designs.

8. Document Revision History for Intel L- and H-tile Avalon Memory-mapped+ IP for PCI Express User Guide

Document Version	Quartus Prime Version	Changes
2024.03.05	23.4	Added additional information for Questa Intel FPGA Edition in *Steps to Run Simulation* table in *Simulating the Design Example* section.
2021.10.19	21.1	Changed the device support level for Stratix 10 to Final Support in the *Device Family Support* section.
2021.05.28	21.1	Added an Appendix chapter on Root Port enumeration. Added a note to the *Features* section stating that L- and H-tile Avalon Memory-mapped+ IP for PCI Express only supports the Separate Reference Clock With No Spread Spectrum architecture (SRNS), but not the Separate Reference Clock With Independent Spread Spectrum architecture (SRIS).
2021.01.11	20.4	Added a figure of a sample descriptor table to the *Programming Model for the Example Design* section.
2020.12.14	20.4	Removed the Multi-function and Function-Level Reset (FLR) parameters from the *Parameters* chapter since these features are not available. Added descriptions for the signals in the Bus Master Enable (BME) conduit to the *System Interfaces* section.
2020.10.05	20.3	Updated the IP name to *Intel L-/H-tile Avalon Memory-mapped+ IP for PCI Express.* Added descriptions for the `tl_cfg_*` signals to the *Configuration Output Interface* section.
2020.06.03	19.4	Added the description for the new input `ninit_done` to the *Clocks and Resets* section. Also added a link to *AN 891: Using the Reset Release Intel FPGA IP*, which describes the Reset Release IP that is used to drive the `ninit_done` input.
2020.04.22	19.3	Updated the document title to *Avalon memory mapped Stratix 10 Hard IP+ for PCI Express Solutions User Guide* to meet new legal naming guidelines. Fixed a typo in the byte address of some Reserved bits in *Table 49. Correspondence between Configuration Space Capability Structures and the PCIe Base Specification Description*.
2019.11.05	19.3	Changed the maximum BAR size to 8 EBytes in the *Base Address Registers* section.
2019.09.30	19.3	Added a note to clarify that this User Guide is applicable to H-Tile and L-Tile variants of the Stratix 10 devices only. Changed the device family support level to Preliminary. Added the Autonomous Hard IP mode to the *Features* list. Updated the Figure *Example of Interrupt Controller Integration with Endpoint Avalon-MM Stratix 10 Hard IP+ for PCIe* to replace the `msi_*` signals with `tl_cfg_*` signals.
2019.07.19	19.1	Added descriptions for `int_status[10:8]` (available for H-Tile only).

continued...

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Document Version	Quartus Prime Version	Changes
2019.07.18	19.1	Added a note stating that `refclk` must be stable and free-running at device power-up for a successful device configuration. Added descriptions for the `int_status[7:0]` and `int_status_common[2:0]` interface signals in the *Hard IP Status and Link Training Conduit* section.
2019.03.30	19.1	Reworded some sections to clarify that this IP core does not include an internal DMA controller. It only contains a BAR Interpreter and Data Movers. The DMA design example does include an external DMA controller. If you are not using the DMA design example, you need to implement your own external DMA controller. Changed the term Descriptor Controller to DMA Controller. Clarified that `bam_response_i[1:0]` are reserved inputs and should be driven to 0.
2019.03.05	18.1.1	Updated the simulation commands in the *Simulating the Design Example* section.
2019.01.16	18.1.1	Removed the section on BIOS enumeration issues because it is not applicable to Stratix 10 devices.
2018.12.24	18.1.1	Added the completion timeout checking feature. Added the PCIe Link Inspector overview.
2018.10.31	18.1	Added the channel layout figure to the *Channel Layout and PLL Usage* section.
2018.09.24	18.1	Added descriptions for two new signal busses, `flr_pf_active_o[<PFNUM> - 1 : 0]` and `flr_pf_done_i[<PFNUM> -1 : 0]`. Updated the steps to run ModelSim simulations for a design example. Updated the steps to run a design example.
2018.08.29	18.0	Added the step to invoke vsim to the instructions for running ModelSim simulations.
2018.05.07	18.0	Initial release.

A. Avalon-MM IP Variants Comparison

Table 65. Differences Between the Intel L-/H-Tile Avalon-MM for PCI Express IP and Intel L-/H-Tile Avalon-MM+ for PCI Express IP

Features	Intel L-/H-Tile Avalon-MM for PCI Express IP	Intel L-/H-Tile Avalon-MM + for PCI Express IP	Comments
PCIe Link Widths	x1/x2/x4/x8	x16	
Lane Rates	2.5/5/8 Gb/s	8 Gb/s Lane Rate	
Root Port Support	Supported	N/A	
Endpoint Support	Supported	Supported	
Data Bus Width on Avalon-MM Interface (Non-Bursting)	32-bit	N/A	
Data Bus Width on Avalon-MM Interface (Bursting)	256-bit	512-bit	
Application Layer clock	250 MHz	250 MHz	
Maximum Payload Size (MPS)	128/256/512 Bytes	128/256/512 Bytes	Default value is 512
Maximum Read Request Size (MRRS)	128/256/512 Bytes	128/256/512 Bytes	Default value is 512
Maximum Outstanding Read Requests (Non-Bursting)	1	N/A	
Maximum Outstanding Read Requests (Bursting)	32	64	
Maximum Burst Size (Non-Bursting)	1 cycle	N/A	
Maximum Burst Size (Bursting)	16 cycles	8 cycles	
Byte Enable Granularity (Non-Bursting)	Byte	N/A	
Byte Enable Granularity HPRXM (Bursting)	Byte	Byte	For single-cycle reads, byte granularity is supported. For multiple-cycle reads, all byte enables are active.
Byte Enable Granularity HPTXS (Bursting)	Dword	Byte	
Write Data Mover	WR_DMA interface	WRDM interface	
Number of Descriptor Queues for Write Data Mover	One	Two	The Intel L-/H-Tile Avalon-MM+ for PCI Express IP has a normal descriptor queue and a priority descriptor queue

continued...

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Features	Intel L-/H-Tile Avalon-MM for PCI Express IP	Intel L-/H-Tile Avalon-MM + for PCI Express IP	Comments
Single Source Address Mode for Write Data Mover	N/A	Supported	Platform Designer Interconnect may increment the address for the Intel L-/H-Tile Avalon-MM+ for PCI Express IP
Read Data Mover	RD_DMA interface	RDDM interface	
Number of Descriptor Queues for Read Data Mover	One	Two	The Intel L-/H-Tile Avalon-MM+ for PCI Express IP has a normal descriptor queue and a priority descriptor queue
Single Destination Address Mode for Read Data Mover	N/A	Supported	Platform Designer Interconnect may increment the address for the Intel L-/H-Tile Avalon-MM+ for PCI Express IP
Avalon-MM Slave	TXS interface	N/A	
Avalon-MM Master	RXM Interface	N/A	
High-Performance Avalon-MM Slave	HPTXS interface	BAS (Bursting Avalon-MM Slave) interface	
High-Performance Avalon-MM Master	HPRXM interface	BAM (Bursting Avalon-MM Master) interface	
Simultaneous support for DMA modules and Avalon-MM masters and slaves	Yes	Yes	
Multi-function Support	N/A	N/A	
CRA (Configuration Register Access)	CRA (Configuration Register Access)	N/A	
CEB (Configuration Extension Bus)	CEB (Configuration Extension Bus)	N/A	
MSI, MSI-X	MSI, MSI-X Interfaces	Not available as separate interfaces. Can be accessed via tl_cfg interface.	Because MSI and MSI-X conduits are not exported by this IP core, the address and data information necessary to send an MSI or MSI-X can be extracted from the tl_cfg interface. Alternatively, that information can be extracted from an immediate write descriptor if you enable that feature by setting bit [146] in the descriptor.
External DMA Controller	Supported	Supported	User has to provide the external DMA controller. The design example contains a DMA controller as an example.
Internal DMA Controller	Supported	N/A	
Number of RX Masters	Up to 6 (one for each BAR)	Single Bursting Master with BAR sideband signals. Supports up to 7 BARs (including expansion ROM BAR)	Multiple BARs are supported by having BAR ID included in a conduit extension of the BAM address

continued...

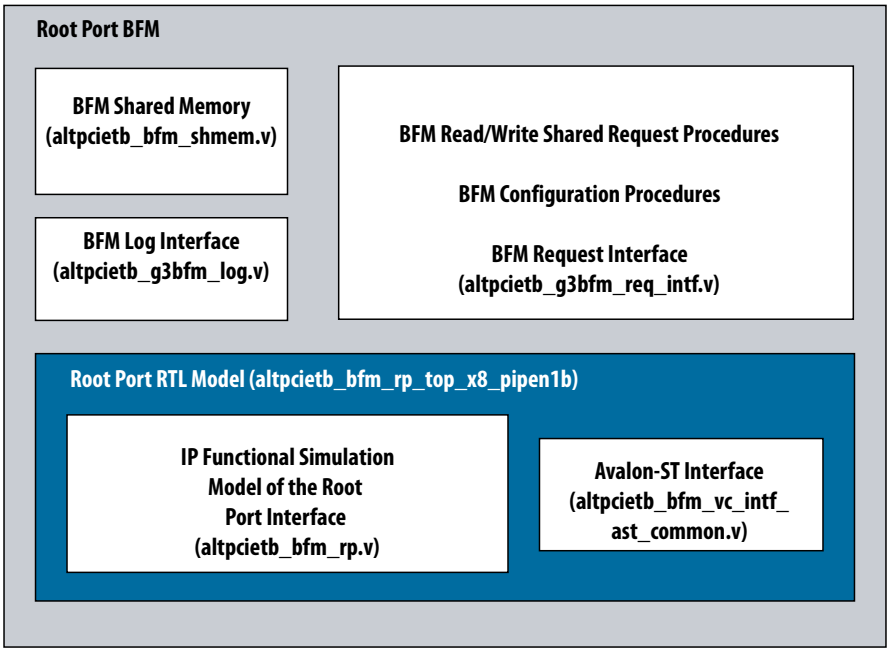
Features	Intel L-/H-Tile Avalon-MM for PCI Express IP	Intel L-/H-Tile Avalon-MM + for PCI Express IP	Comments
TPH (TLP Processing Hint)	N/A	N/A	
ATS (Address Translation Service)	N/A	N/A	
Error Handling	N/A	N/A	
AER (Advanced Error Reporting)	Supported (always enabled)	Supported (always enabled)	
Hard IP Reconfiguration	HIP_RECONFIG interface	HIP_RECONFIG interface	
XCVR Reconfiguration	XCVR_RECONFIG interface	XCVR_RECONFIG interface	
FPLL Reconfiguration	RECONFIG_PLL0 interface	RECONFIG_PLL0 interface	
LC PLL Reconfiguration	RECONFIG_PLL1 interface	RECONFIG_PLL1 interface	
Support for PCIe Link Inspector	Supported	N/A	It is supported up to Gen3x8 for both Avalon-ST and Avalon-MM. It is not yet supported in Gen3x16.
Design Example Availability	Yes	Yes	
Software Programming Model	Single descriptor queue in Data Movers, so no prioritization of simultaneous DMA transactions.	Better prioritization of simultaneous DMA transactions due to the different descriptor queues. Improved interrupt generation.	

B. Root Port BFM

B.1. Root Port BFM Overview

The basic Root Port BFM provides a Verilog HDL task-based interface to test the PCIe link. The Root Port BFM also handles requests received from the PCIe link. The following figure provides an overview of the Root Port BFM.

Figure 29. Root Port BFM



Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

The following descriptions provides an overview of the blocks shown in the *Root Port BFM* figure:

- BFM shared memory (`altpcietb_g3bfm_shmem.v`): The BFM memory performs the following tasks:
  - Stores data received with all completions from the PCI Express link.
  - Stores data received with all write transactions received from the PCI Express link.
  - Sources data for all completions in response to read transactions received from the PCI Express link.
  - Sources data for most write transactions issued to the link. The only exception is certain BFM PCI Express write procedures that have a four-byte field of write data passed in the call.
  - Stores a data structure that contains the sizes of and the values programmed in the BARs of the Endpoint.

A set of procedures read, write, fill, and check the shared memory from the BFM driver. For details on these procedures, see *BFM Shared Memory Access Procedures*.

- BFM Read/Write Request Functions (`altpcietb_g3bfm_rdwr.v`): These functions provide the basic BFM calls for PCI Express read and write requests. For details on these procedures, refer to *BFM Read and Write Procedures*.
- BFM Configuration Functions (`altpcietb_g3bfm_rp.v`): These functions provide the BFM calls to request configuration of the PCI Express link and the Endpoint Configuration Space registers. For details on these procedures and functions, refer to *BFM Configuration Procedures*.
- BFM Log Interface (`altpcietb_g3bfm_log.v`): The BFM log functions provides routines for writing commonly formatted messages to the simulator standard output and optionally to a log file. It also provides controls that stop simulation on errors. For details on these procedures, refer to *BFM Log and Message Procedures*.
- BFM Request Interface (`altpcietb_g3bfm_req_intf.v`): This interface provides the low-level interface between the `altpcietb_g3bfm_rdwr.v` and `altpcietb_g3bfm_configure.v` procedures or functions and the Root Port RTL Model. This interface stores a write-protected data structure containing the sizes and the values programmed in the BAR registers of the Endpoint. This interface also stores other critical data used for internal BFM management. You do not need to access these files directly to adapt the testbench to test your Endpoint application.
- Avalon-ST Interfaces (`altpcietb_bfm_vc_intf_ast_common.v`): These interface modules handle the Root Port interface model. They take requests from the BFM request interface and generate the required PCI Express transactions. They handle completions received from the PCI Express link and notify the BFM request interface when requests are complete. Additionally, they handle any requests received from the PCI Express link, and store or fetch data from the shared memory before generating the required completions.

B.2. Issuing Read and Write Transactions to the Application Layer

The `ebfm_bar` procedures in `altpcietb_bfm_rdwr.v` implement read and write transactions to the Endpoint Application Layer. The procedures and functions listed below are available in the Verilog HDL include file `altpcietb_bfm_rdwr.v`.

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

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

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

B.3. Configuration of Root Port and Endpoint

Before you issue transactions to the Endpoint, you must configure the Root Port and Endpoint Configuration Space registers.

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

1. Sets the Root Port Configuration Space to enable the Root Port to send transactions on the PCI Express link.
2. Sets the Root Port and Endpoint PCI Express Capability Device Control registers as follows:
   a. Disables `Error Reporting` in both the Root Port and Endpoint. The BFM does not have error handling capability.
   b. Enables `Relaxed Ordering` in both Root Port and Endpoint.
   c. Enables `Extended Tags` for the Endpoint if the Endpoint has that capability.
   d. Disables `Phantom Functions`, `Aux Power PM`, and `No Snoop` in both the Root Port and Endpoint.

e. Sets the `Max Payload Size` to the value that the Endpoint supports because the Root Port supports the maximum payload size.

f. Sets the Root Port `Max Read Request Size` to 4 KB because the example Endpoint design supports breaking the read into as many completions as necessary.

g. Sets the Endpoint `Max Read Request Size` equal to the `Max Payload Size` because the Root Port does not support breaking the read request into multiple completions.

3. Assigns values to all the Endpoint BAR registers. The BAR addresses are assigned by the algorithm outlined below.

   a. I/O BARs are assigned smallest to largest starting just above the ending address of BFM shared memory in I/O space and continuing as needed throughout a full 32-bit I/O space.

   b. The 32-bit non-prefetchable memory BARs are assigned smallest to largest, starting just above the ending address of BFM shared memory in memory space and continuing as needed throughout a full 32-bit memory space.

   c. The value of the `addr_map_4GB_limit` input to the `ebfm_cfg_rp_ep` procedure controls the assignment of the 32-bit prefetchable and 64-bit prefetchable memory BARS. The default value of `addr_map_4GB_limit` is `0`.

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

      However, if `addr_map_4GB_limit` is set to 1, the address map is limited to 4 GB. The `ebfm_cfg_rp_ep` procedure assigns 32-bit and 64-bit prefetchable memory BARs largest to smallest, starting at the 4 GB address and continuing as needed down to the ending address of the last 32-bit non-prefetchable BAR.

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

4. Based on the above BAR assignments, the `ebfm_cfg_rp_ep` procedure assigns the Root Port Configuration Space address windows to encompass the valid BAR address ranges.

5. The `ebfm_cfg_rp_ep` procedure enables master transactions, memory address decoding, and I/O address decoding in the Endpoint PCIe control register.

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

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

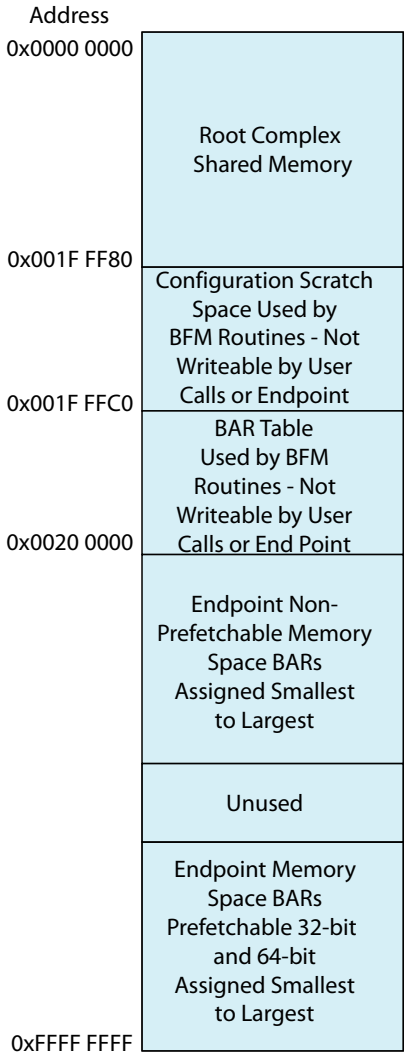
Table 66. BAR Table Structure

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

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

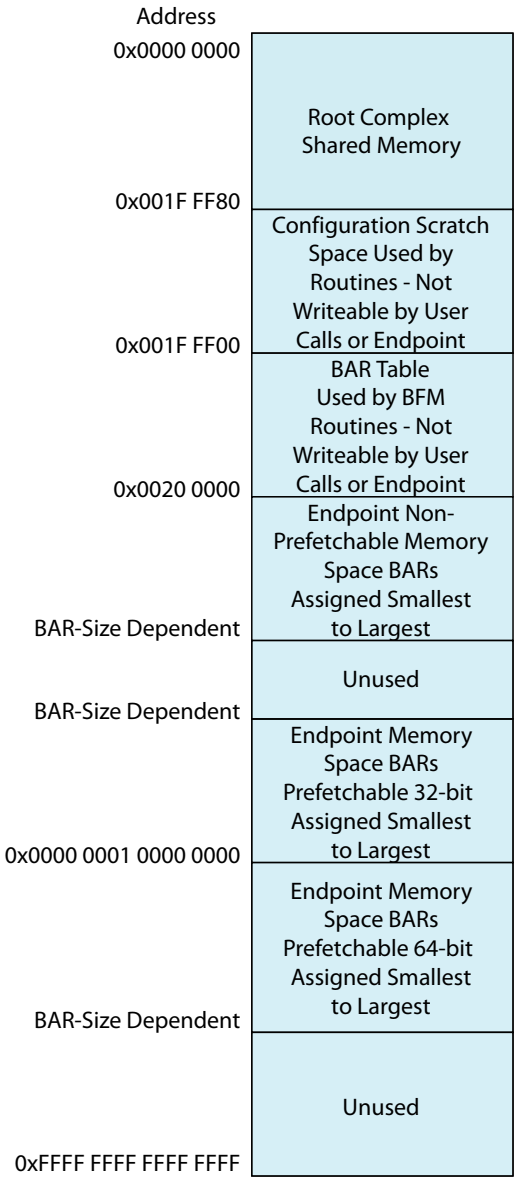
Besides the `ebfm_cfg_rp_ep` procedure in `altpcietb_bfm_rp_gen3_x8.sv`, routines to read and write Endpoint Configuration Space registers directly are available in the Verilog HDL include file. After the `ebfm_cfg_rp_ep` procedure runs the PCI Express I/O and Memory Spaces have the layout shown in the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

Figure 30. Memory Space Layout—4 GB Limit



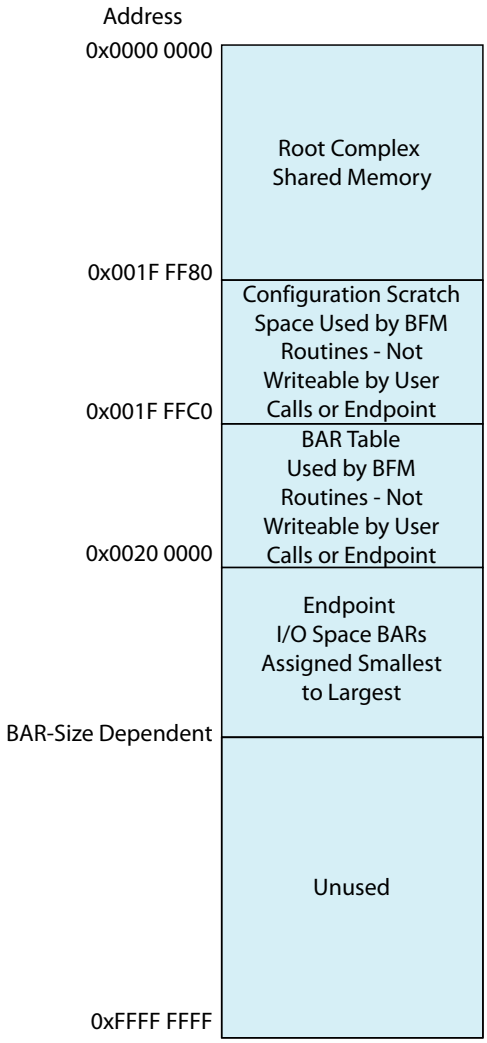
The following figure shows the resulting memory space map when the **addr_map_4GB_limit** is 0.

Figure 31. Memory Space Layout—No Limit



The following figure shows the I/O address space.

Figure 32. I/O Address Space



B.4. Configuration Space Bus and Device Numbering

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

B.5. BFM Memory Map

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

C. BFM Procedures and Functions

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

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

C.1. ebfm_barwr Procedure

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

Location	altpcietb_g3bfm_rdwr.v	
Syntax	`ebfm_barwr(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory. The `bar_table` structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	`bar_num`	Number of the BAR used with `pcie_offset` to determine PCI Express address.
	`pcie_offset`	Address offset from the BAR base.
	`lcladdr`	BFM shared memory address of the data to be written.
	`byte_len`	Length, in bytes, of the data written. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	`tclass`	Traffic class used for the PCI Express transaction.

C.2. ebfm_barwr_imm Procedure

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

Location	altpcietb_g3bfm_rdwr.v	
Syntax	`ebfm_barwr_imm(bar_table, bar_num, pcie_offset, imm_data, byte_len, tclass)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory. The `bar_table` structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.

continued...

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Location	altpcietb_g3bfm_rdwr.v	
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.
	pcie_offset	Address offset from the BAR base.
	imm_data	Data to be written. In Verilog HDL, this argument is reg [31:0].In both languages, the bits written depend on the length as follows: Length Bits Written • 4: 31 down to 0 • 3: 23 down to 0 • 2: 15 down to 0 • 1: 7 down to 0
	byte_len	Length of the data to be written in bytes. Maximum length is 4 bytes.
	tclass	Traffic class to be used for the PCI Express transaction.

C.3. ebfm_barrd_wait Procedure

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

Location	altpcietb_g3bfm_rdwr.v	
Syntax	ebfm_barrd_wait(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory. The bar_table structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.
	pcie_offset	Address offset from the BAR base.
	lcladdr	BFM shared memory address where the read data is stored.
	byte_len	Length, in bytes, of the data to be read. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	tclass	Traffic class used for the PCI Express transaction.

C.4. ebfm_barrd_nowt Procedure

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

Location	altpcietb_g3bfm_rdwr.v	
Syntax	`ebfm_barrd_nowt(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	Number of the BAR used with `pcie_offset` to determine PCI Express address.
	`pcie_offset`	Address offset from the BAR base.
	`lcladdr`	BFM shared memory address where the read data is stored.
	`byte_len`	Length, in bytes, of the data to be read. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	`tclass`	Traffic Class to be used for the PCI Express transaction.

C.5. ebfm_cfgwr_imm_wait Procedure

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

Location	altpcietb_g3bfm_rdwr.v	
Syntax	`ebfm_cfgwr_imm_wait(bus_num, dev_num, fnc_num, imm_regb_ad, regb_ln, imm_data, compl_status`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data written. Maximum length is four bytes. The `regb_ln` and the `regb_ad` arguments cannot cross a DWORD boundary.
	`imm_data`	Data to be written. This argument is `reg [31:0]`. The bits written depend on the length: • 4: 31 down to 0 • 3: 23 down to 0 • 2: 15 down to 0 • 1: 7 down to 0
	`compl_status`	This argument is `reg [2:0]`. This argument is the completion status as specified in the PCI Express specification. The following encodings are defined: • 3'b000: SC— Successful completion • 3'b001: UR— Unsupported Request • 3'b010: CRS — Configuration Request Retry Status • 3'b100: CA — Completer Abort

C.6. ebfm_cfgwr_imm_nowt Procedure

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

Location	altpcietb_g3bfm_rdwr.v	
Syntax	`ebfm_cfgwr_imm_nowt(bus_num, dev_num, fnc_num, imm_regb_adr, regb_len, imm_data)`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data written. Maximum length is four bytes, The `regb_ln` the `regb_ad` arguments cannot cross a DWORD boundary.
	`imm_data`	Data to be written This argument is `reg [31:0]`. In both languages, the bits written depend on the length. The following encodes are defined. • 4: [31:0] • 3: [23:0] • 2: [15:0] • 1: [7:0]

C.7. ebfm_cfgrd_wait Procedure

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

Location	altpcietb_g3bfm_rdwr.v	
Syntax	`ebfm_cfgrd_wait(bus_num, dev_num, fnc_num, regb_ad, regb_ln, lcladdr, compl_status)`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data read. Maximum length is four bytes. The `regb_ln` and the `regb_ad` arguments cannot cross a DWORD boundary.
	`lcladdr`	BFM shared memory address of where the read data should be placed.
	`compl_status`	Completion status for the configuration transaction. This argument is reg [2:0].

continued...

Location	altpcietb_g3bfm_rdwr.v	
		This is the completion status as specified in the PCI Express specification. The following encodings are defined. • 3'b000: SC— Successful completion • 3'b001: UR— Unsupported Request • 3'b010: CRS — Configuration Request Retry Status • 3'b100: CA — Completer Abort

C.8. ebfm_cfgrd_nowt Procedure

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

Location	altpcietb_g3bfm_rdwr.v	
Syntax	`ebfm_cfgrd_nowt(bus_num, dev_num, fnc_num, regb_ad, regb_ln, lcladdr)`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data written. Maximum length is four bytes. The `regb_ln` and `regb_ad` arguments cannot cross a DWORD boundary.
	`lcladdr`	BFM shared memory address where the read data should be placed.

C.9. BFM Configuration Procedures

The BFM configuration procedures are available in `altpcietb_bfm_rp_gen3_x8.sv` . These procedures support configuration of the Root Port and Endpoint Configuration Space registers.

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

C.9.1. ebfm_cfg_rp_ep Procedure

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

Location	altpcietb_g3bfm_configure.v	
Syntax	`ebfm_cfg_rp_ep(bar_table, ep_bus_num, ep_dev_num, rp_max_rd_req_size, display_ep_config, addr_map_4GB_limit)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory. This routine populates the `bar_table` structure. The `bar_table` structure stores the size of each BAR and the address values assigned to

continued...

Location	altpcietb_g3bfm_configure.v	
		each BAR. The address of the bar_table structure is passed to all subsequent read and write procedure calls that access an offset from a particular BAR.
	ep_bus_num	PCI Express bus number of the target device. This number can be any value greater than 0. The Root Port uses this as the secondary bus number.
	ep_dev_num	PCI Express device number of the target device. This number can be any value. The Endpoint is automatically assigned this value when it receives the first configuration transaction.
	rp_max_rd_req_size	Maximum read request size in bytes for reads issued by the Root Port. This parameter must be set to the maximum value supported by the Endpoint Application Layer. If the Application Layer only supports reads of the MAXIMUM_PAYLOAD_SIZE, then this can be set to 0 and the read request size is set to the maximum payload size. Valid values for this argument are 0, 128, 256, 512, 1,024, 2,048 and 4,096.
	display_ep_config	When set to 1 many of the Endpoint Configuration Space registers are displayed after they have been initialized, causing some additional reads of registers that are not normally accessed during the configuration process such as the Device ID and Vendor ID.
	addr_map_4GB_limit	When set to 1 the address map of the simulation system is limited to 4 GB. Any 64-bit BARs are assigned below the 4 GB limit.

C.9.2. ebfm_cfg_decode_bar Procedure

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

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

C.10. BFM Shared Memory Access Procedures

These procedures and functions support accessing the BFM shared memory.

C.10.1. Shared Memory Constants

The following constants are defined in altrpcietb_g3bfm_shmem.v. They select a data pattern for the shmem_fill and shmem_chk_ok routines. These shared memory constants are all Verilog HDL type integer.

Table 67. Constants: Verilog HDL Type INTEGER

Constant	Description
SHMEM_FILL_ZEROS	Specifies a data pattern of all zeros
SHMEM_FILL_BYTE_INC	Specifies a data pattern of incrementing 8-bit bytes (0x00, 0x01, 0x02, etc.)
SHMEM_FILL_WORD_INC	Specifies a data pattern of incrementing 16-bit words (0x0000, 0x0001, 0x0002, etc.)
SHMEM_FILL_DWORD_INC	Specifies a data pattern of incrementing 32-bit DWORDs (0x00000000, 0x00000001, 0x00000002, etc.)
SHMEM_FILL_QWORD_INC	Specifies a data pattern of incrementing 64-bit qwords (0x0000000000000000, 0x0000000000000001, 0x0000000000000002, etc.)
SHMEM_FILL_ONE	Specifies a data pattern of all ones

Related Information

ebfm_display Verilog HDL Function on page 104

C.10.2. shmem_write Procedure

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

Location	altpcietb_g3bfm_shmem.v	
Syntax	`shmem_write(addr, data, leng)`	
Arguments	`addr`	BFM shared memory starting address for writing data
	`data`	Data to write to BFM shared memory. This parameter is implemented as a 64-bit vector. `leng` is 1–8 bytes. Bits 7 down to 0 are written to the location specified by `addr`; bits 15 down to 8 are written to the `addr+1` location, etc.
	`length`	Length, in bytes, of data written

C.10.3. shmem_read Function

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

Location	altpcietb_g3bfm_shmem.v	
Syntax	`data:= shmem_read(addr, leng)`	
Arguments	`addr`	BFM shared memory starting address for reading data
	`leng`	Length, in bytes, of data read
Return	`data`	Data read from BFM shared memory. This parameter is implemented as a 64-bit vector. `leng` is 1- 8 bytes. If `leng` is less than 8 bytes, only the corresponding least significant bits of the returned data are valid. Bits 7 down to 0 are read from the location specified by `addr`; bits 15 down to 8 are read from the addr+1 location, etc.

C.10.4. shmem_display Verilog HDL Function

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

Location	altrpcietb_g3bfm_shmem.v	
Syntax	Verilog HDL: `dummy_return:=shmem_display(addr, leng, word_size, flag_addr, msg_type);`	
Arguments	`addr`	BFM shared memory starting address for displaying data.
	`leng`	Length, in bytes, of data to display.
	`word_size`	Size of the words to display. Groups individual bytes into words. Valid values are 1, 2, 4, and 8.
	`flag_addr`	Adds a <== flag to the end of the display line containing this address. Useful for marking specific data. Set to a value greater than 2**21 (size of BFM shared memory) to suppress the flag.
	`msg_type`	Specifies the message type to be displayed at the beginning of each line. See "BFM Log and Message Procedures" on page 18–37 for more information about message types. Set to one of the constants defined in Table 18–36 on page 18–41.

C.10.5. shmem_fill Procedure

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

Location	altrpcietb_g3bfm_shmem.v	
Syntax	`shmem_fill(addr, mode, leng, init)`	
Arguments	`addr`	BFM shared memory starting address for filling data.
	`mode`	Data pattern used for filling the data. Should be one of the constants defined in section *Shared Memory Constants*.
	`leng`	Length, in bytes, of data to fill. If the length is not a multiple of the incrementing data pattern width, then the last data pattern is truncated to fit.
	`init`	Initial data value used for incrementing data pattern modes. This argument is `reg [63:0]`. The necessary least significant bits are used for the data patterns that are smaller than 64 bits.

C.10.6. shmem_chk_ok Function

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

Location	altrpcietb_g3bfm_shmem.v	
Syntax	`result:= shmem_chk_ok(addr, mode, leng, init, display_error)`	
Arguments	`addr`	BFM shared memory starting address for checking data.
	`mode`	Data pattern used for checking the data. Should be one of the constants defined in section "Shared Memory Constants" on page 18–35.
	`leng`	Length, in bytes, of data to check.

continued...

Location	altrpcietb_g3bfm_shmem.v	
	`init`	This argument is `reg [63:0]`.The necessary least significant bits are used for the data patterns that are smaller than 64-bits.
	`display_error`	When set to 1, this argument displays the data failing comparison on the simulator standard output.
Return	`Result`	Result is 1-bit. • 1'b1 — Data patterns compared successfully • 1'b0 — Data patterns did not compare successfully

C.11. BFM Log and Message Procedures

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

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

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

Certain message types stop the simulation after the message is displayed. The following table shows the default value determining whether a message type stops the simulation. You can specify whether the simulation stops for particular messages with the procedure `ebfm_log_set_stop_on_msg_mask`. You can also suppress the display of certain message types. The following table also shows the default values determining whether a message type is displayed. To change the default message display, modify the display default value with a procedure call to `ebfm_log_set_suppressed_msg_mask`.

All of these log message constants type `integer`.

Table 68. Log Messages

Constant (Message Type)	Description	Mask Bit No	Display by Default	Simulation Stops by Default	Message Prefix
`EBFM_MSG_DEBUG`	Specifies debug messages.	0	No	No	`DEBUG:`
`EBFM_MSG_INFO`	Specifies informational messages, such as configuration register values, starting and ending of tests.	1	Yes	No	`INFO:`
`EBFM_MSG_WARNING`	Specifies warning messages, such as tests being skipped due to the specific configuration.	2	Yes	No	`WARNING:`
`EBFM_MSG_ERROR_INFO`	Specifies additional information for an error. Use this message to display preliminary information before an error message that stops simulation.	3	Yes	No	`ERROR:`

continued...

Constant (Message Type)	Description	Mask Bit No	Display by Default	Simulation Stops by Default	Message Prefix
EBFM_MSG_ERROR_CONTINUE	Specifies a recoverable error that allows simulation to continue. Use this error for data comparison failures.	4	Yes	No	ERROR:
EBFM_MSG_ERROR_FATAL	Specifies an error that stops simulation because the error leaves the testbench in a state where further simulation is not possible.	N/A	Yes Cannot suppress	Yes Cannot suppress	FATAL:
EBFM_MSG_ERROR_FATAL_TB_ERR	Used for BFM test driver or Root Port BFM fatal errors. Specifies an error that stops simulation because the error leaves the testbench in a state where further simulation is not possible. Use this error message for errors that occur due to a problem in the BFM test driver module or the Root Port BFM, that are not caused by the Endpoint Application Layer being tested.	N/A	Y Cannot suppress	Y Cannot suppress	FATAL:

C.11.1. ebfm_display Verilog HDL Function

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

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

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

Location	altrpcietb_g3bfm_log.v	
Syntax	`Verilog HDL: dummy_return:=ebfm_display(msg_type, message);`	
Argument	`msg_type`	Message type for the message. Should be one of the constants defined in *Shared Memory Constants*.
	`message`	The message string is limited to a maximum of 100 characters. Also, because Verilog HDL does not allow variable length strings, this routine strips off leading characters of 8'h00 before displaying the message.
Return	`always 0`	Applies only to the Verilog HDL routine.

Related Information

Shared Memory Constants on page 100

C.11.2. ebfm_log_stop_sim Verilog HDL Function

The `ebfm_log_stop_sim` procedure stops the simulation.

Location	altrpcietb_g3bfm_log.v	
Syntax	Verilog HDL: `return:=ebfm_log_stop_sim(success);`	
Argument	`success`	When set to a 1, this process stops the simulation with a message indicating successful completion. The message is prefixed with SUCCESS. Otherwise, this process stops the simulation with a message indicating unsuccessful completion. The message is prefixed with FAILURE.
Return	Always 0	This value applies only to the Verilog HDL function.

C.11.3. ebfm_log_set_suppressed_msg_mask Task

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

Location	altrpcietb_g3bfm_log.v	
Syntax	`ebfm_log_set_suppressed_msg_mask (msg_mask)`	
Argument	`msg_mask`	This argument is `reg [EBFM_MSG_ERROR_CONTINUE: EBFM_MSG_DEBUG]`. A 1 in a specific bit position of the `msg_mask` causes messages of the type corresponding to the bit position to be suppressed.

C.11.4. ebfm_log_set_stop_on_msg_mask Verilog HDL Task

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

Location	altrpcietb_g3bfm_log.v	
Syntax	`ebfm_log_set_stop_on_msg_mask (msg_mask)`	
Argument	`msg_mask`	This argument is `reg [EBFM_MSG_ERROR_CONTINUE:EBFM_MSG_DEBUG]`. A 1 in a specific bit position of the `msg_mask` causes messages of the type corresponding to the bit position to stop the simulation after the message is displayed.

C.11.5. ebfm_log_open Verilog HDL Function

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

Location	altrpcietb_g3bfm_log.v	
Syntax	`ebfm_log_open (fn)`	
Argument	`fn`	This argument is type `string` and provides the file name of log file to be opened.

C.11.6. ebfm_log_close Verilog HDL Function

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

Location	altrpcietb_g3bfm_log.v
Syntax	ebfm_log_close
Argument	NONE

C.12. Verilog HDL Formatting Functions

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

C.12.1. himage1

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

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

C.12.2. himage2

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

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

C.12.3. himage4

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

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

C.12.4. himage8

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

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

C.12.5. himage16

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

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

C.12.6. dimage1

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

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

C.12.7. dimage2

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

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

C.12.8. dimage3

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

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

C.12.9. dimage4

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

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

C.12.10. dimage5

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

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

C.12.11. dimage6

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

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

C.12.12. dimage7

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

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

D. Troubleshooting and Observing the Link Status

D.1. Troubleshooting

Provides a table listing common problems and solutions. If the IP code supports the Transceiver Toolkit, refer to the Transceiver Toolkit Web Page for issues with transceiver link signal integrity.

Example

Table 69. Link Hangs in L0

Possible Causes	Symptoms and Root Causes	Workarounds and Solutions
Avalon-ST signaling violates Avalon-ST protocol	Avalon-ST protocol violations include the following errors: • More than one `tx_st_sop` per `tx_st_eop`. • Two or more `tx_st_eop`'s without a corresponding `tx_st_sop`. • `rx_st_valid` is not asserted with `tx_st_sop` or `tx_st_eop`. These errors are applicable to both simulation and hardware.	Add logic to detect situations where `tx_st_ready` remains deasserted for more than 100 cycles. Set post-triggering conditions to check for the Avalon-ST signaling of last two TLPs to verify correct `tx_st_sop` and `tx_st_eop` signaling.
Incorrect payload size	Determine if the length field of the last TLP transmitted by End Point is greater than the InitFC credit advertised by the link partner. For simulation, refer to the log file and simulation dump. For hardware, use a third-party logic analyzer trace to capture PCIe transactions.	If the payload is greater than the initFC credit advertised, you must either increase the InitFC of the posted request to be greater than the **max payload size** or reduce the payload size of the requested TLP to be less than the InitFC value.
Flow control credit overflows	Determine if the credit field associated with the current TLP type in the `tx_cred` bus is less than the requested credit value. When insufficient credits are available, the core waits for the link partner to release the correct credit type. Sufficient credits may be unavailable if the link partner increments credits more than expected, creating a situation where the Arria 10 Hard IP for PCI Express IP Core credit calculation is out-of-sync with its link partner.	Add logic to detect conditions where the `tx_st_ready` signal remains deasserted for more than 100 cycles. Set post-triggering conditions to check the value of the `tx_cred_*` and `tx_st_*` interfaces. Add a FIFO status signal to determine if the TXFIFO is full.
Malformed TLP is transmitted	Refer to the error log file to find the last good packet transmitted on the link. Correlate this packet with TLP sent on Avalon-ST	Revise the Application Layer logic to correct the error condition.

continued...

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Possible Causes	Symptoms and Root Causes	Workarounds and Solutions
	interface. Determine if the last TLP sent has any of the following errors: • The actual payload sent does not match the length field. • The format and type fields are incorrectly specified. • TD field is asserted, indicating the presence of a TLP digest (ECRC), but the ECRC dword is not present at the end of TLP. • The payload crosses a 4KByte boundary.	
Insufficient Posted credits released by Root Port	If a Memory Write TLP is transmitted with a payload greater than the **maximum payload size**, the Root Port may release an incorrect posted data credit to the Endpoint in simulation. As a result, the Endpoint does not have enough credits to send additional Memory Write Requests.	Make sure Application Layer sends Memory Write Requests with a payload less than or equal the value specified by the **maximum payload size**.
Missing completion packets or dropped packets	The RX Completion TLP might cause the RX FIFO to overflow. Make sure that the total outstanding read data of all pending Memory Read Requests is smaller than the allocated completion credits in RX buffer.	You must ensure that the data for all outstanding read requests does not exceed the completion credits in the RX buffer.

Related Information

Transceiver Toolkit

D.1.1. Simulation Fails to Progress Beyond Polling.Active State

If your PIPE simulation cycles between the Detect.Quiet (6'h00), Detect.Active (6'h01), and Polling.Active (6'h02) LTSSM states, the PIPE interface width may be incorrect. The width of the DUT top-level PIPE interface is 32 bits for Stratix 10 devices.

Table 70. Changes for 32-Bit PIPE Interface

8-Bit PIPE Interface	32-Bit PIPE Interface
`output wire [7:0] pcie_s10_hip_0_hip_pipe_txdata0`	`output wire [31:0] pcie_s10_hip_0_hip_pipe_txdata0`
`input wire [7:0] pcie_s10_hip_0_hip_pipe_rxdata0`	`input wire [31:0] pcie_s10_hip_0_hip_pipe_rxdata0`
`output wire pcie_s10_simulation_inst_pcie_s10_hip_0_hip_pipe_txdatak0`	`output wire [3:0] pcie_s10_simulation_inst_pcie_s10_hip_0_hip_pipe_txdatak0`
`input wire pcie_s10_simulation_inst_pcie_s10_hip_0_hip_pipe_rxdatak0`	`input wire [3:0] pcie_s10_simulation_inst_pcie_s10_hip_0_hip_pipe_rxdatak0`

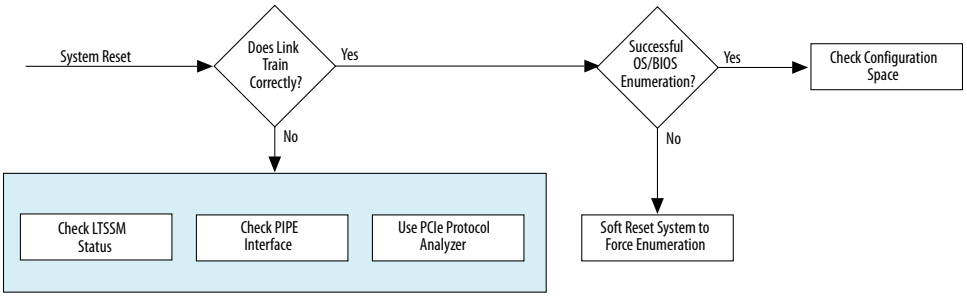
D.1.2. Hardware Bring-Up Issues

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

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

The following sections describe how to debug the hardware bring-up flow. Intel recommends a systematic approach to diagnosing bring-up issues as illustrated in the following figure.

Figure 33. Debugging Link Training Issues



D.1.3. Link Training

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

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

You can use Signal Tap Embedded Logic Analyzer to diagnose the LTSSM state transitions that are occurring on the PIPE interface. The `ltssmstate` bus encodes the status of LTSSM. The LTSSM state machine reflects the Physical Layer's progress through the link training process. For a complete description of the states these signals encode, refer to *Reset, Status, and Link Training Signals*. When link training completes successfully and the link is up, the LTSSM should remain stable in the L0 state. When link issues occur, you can monitor `ltssmstate` to determine the cause.

D.1.4. Use Third-Party PCIe Analyzer

A third-party protocol analyzer for PCI Express records the traffic on the physical link and decodes traffic, saving you the trouble of translating the symbols yourself. A third-party protocol analyzer can show the two-way traffic at different levels for different requirements.

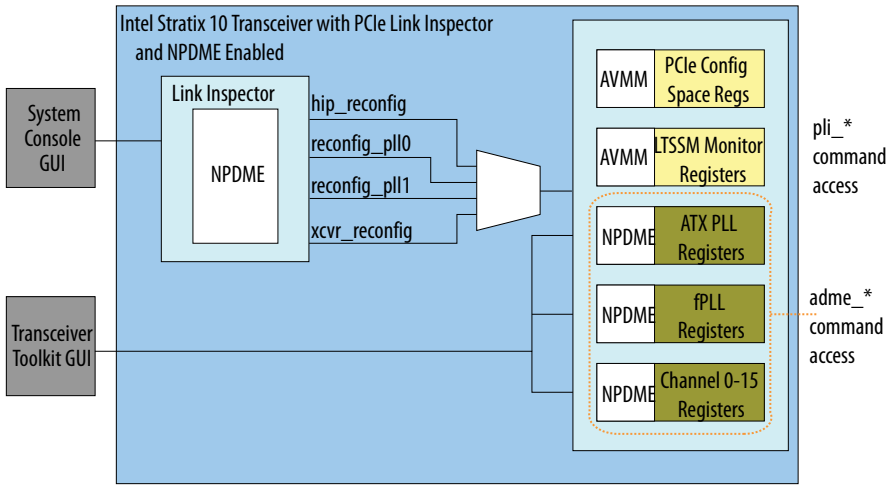
For high-level diagnostics, the analyzer shows the LTSSM flows for devices on both side of the link side-by-side. This display can help you see the link training handshake behavior and identify where the traffic gets stuck. A traffic analyzer can display the contents of packets so that you can verify the contents. For complete details, refer to the third-party documentation.

D.2. PCIe Link Inspector Overview

Use the PCIe Link Inspector to monitor the PCIe link at the Physical, Data Link and Transaction Layers.

The following figure provides an overview of the debug capability available when you enable all of the options on the **Configuration, Debug and Extension Option** tab of the Intel L-/H-Tile Avalon-ST for PCI Express IP component GUI.

Figure 34. Overview of PCIe Link Inspector Hardware



As this figure illustrates, the PCIe Link (`pli*`) commands provide access to the following registers:

- The PCI Express Configuration Space registers
- LTSSM monitor registers
- ATX PLL dynamic partial reconfiguration I/O (DPRIO) registers from the dynamic reconfiguration interface
- fPLL DPRIO registers from the dynamic reconfiguration interface
- Native PHY DPRIO registers from the dynamic reconfiguration interface

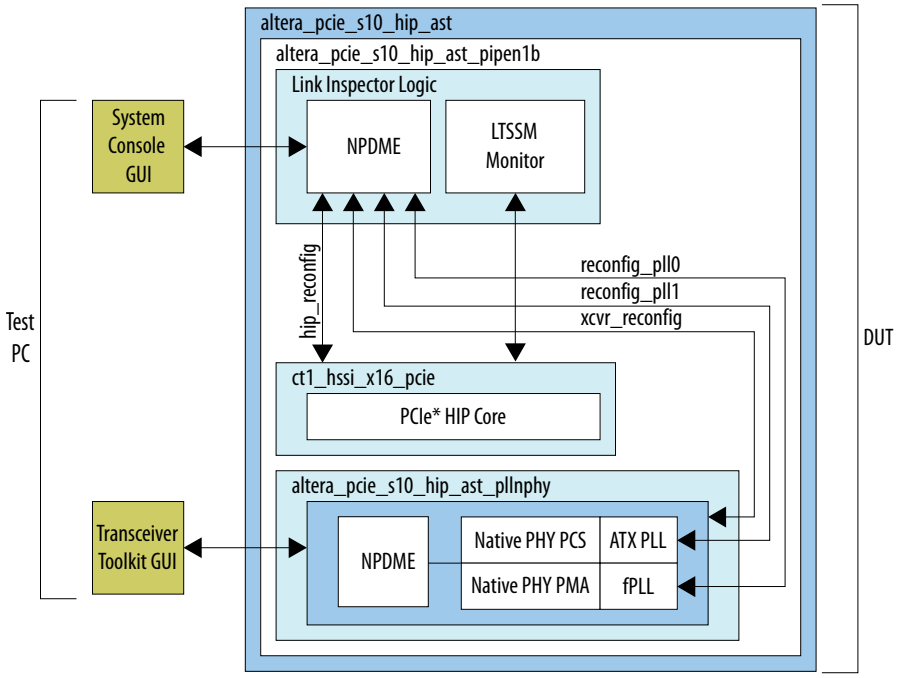
The NPDME commands (currently called `adme*_`) provide access to the following registers

- ATX PLL DPRIO registers from the ATX PLL NPDME interface
- fPLL DPRIO registers from the fPLL NPDME interface
- Native PHY DPRIO registers from the Native PHY NPDME interface

D.2.1. PCIe Link Inspector Hardware

When you enable, the PCIe Link Inspector, the `altera_pcie_s10_hip_ast_pipen1b` module of the generated IP includes the PCIe Link Inspector as shown in the figure below.

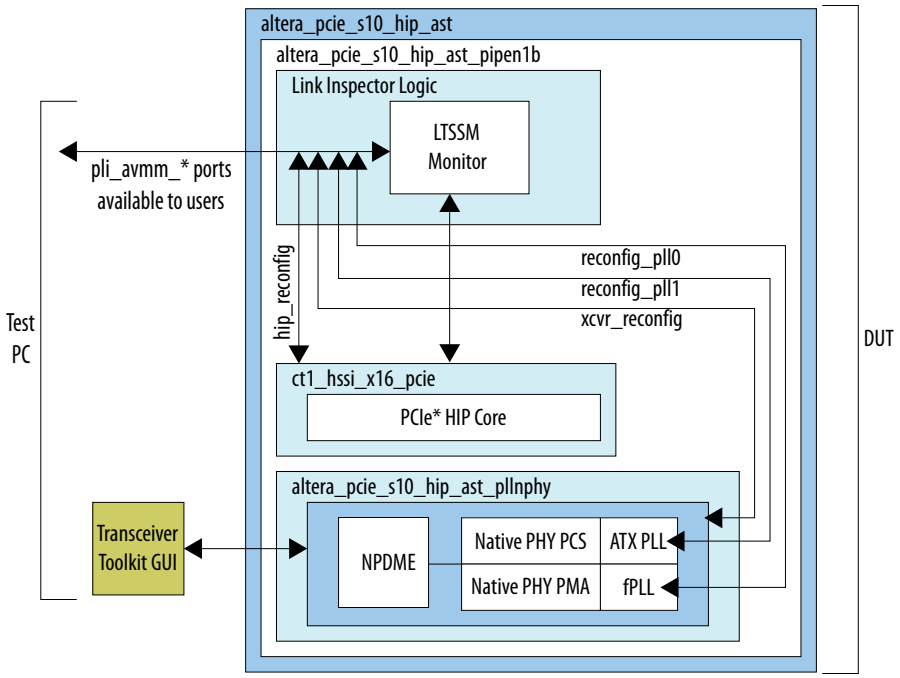
Figure 35. Intel L-/H-Tile Avalon-ST for PCI Express IP with the PCIe Link Inspector



You drive the PCIe Link Inspector from a System Console running on a separate test PC. The System Console connects to the PCIe Link Inspector via a Native PHY Debug Master Endpoint (NPDME). An Intel FPGA Download Cable makes this connection.

You can also access low-level link status information from the PCIe Hard IP, XCVR or PLL blocks via the Link Inspector Avalon-MM Interface by enabling the **Enable PCIe Link Inspector Avalon-MM Interface** option in the IP GUI. See the section *Enabling the Link Inspector* for more details. When you enable this option, you do not need to use the System Console. The `pli_avmm_*` ports that are exposed connect directly to the LTSSM Monitor without going through an NPDME block.

Figure 36. Intel L-/H-Tile Avalon-ST for PCI Express IP with the PCIe Link Inspector and the Link Inspector Avalon-MM Interface Enabled



Note: To use the PCIe Link Inspector, enable the Hard IP dynamic reconfiguration and Transceiver dynamic reconfiguration along with the Link Inspector itself. As a result, the IP exports four clocks (`hip_reconfig_clk`, `xcvr_reconfig_clk`, `reconfig_pll0_clk` and `reconfig_pll1_clk`) and four resets (`hip_reconfig_rst_n`, `xcvr_reconfig_reset`, `reconfig_pll0_reset` and `reconfig_pll1_reset`) to the IP block symbol. These signals provide the clocks and resets to the following interfaces:

- The NPDME module
- FPLL reconfiguration interface (`reconfig_pll0`)
- ATXPLL reconfiguration interface (`reconfig_pll1`)
- Transceiver reconfiguration interface (`xcvr_reconfig`)
- Hard IP reconfiguration interface (`hip_reconfig`)

When you run a dynamically-generated design example on the Stratix 10-GX Development Kit, these signals are automatically connected.

If you run the PCIe Link Inspector on your own hardware, be sure to connect the four clocks mentioned above to a clock source of up to 100 MHz. Additionally, ensure that the four resets mentioned above are connected to an appropriate reset signal.

When you generate a PCIe design example (with a PCIe IP instantiated) without enabling the Link Inspector, the following interfaces are not exposed at the top level of the PCIe IP:

- FPLL reconfiguration interface (`reconfig_pll0`)
- ATXPLL reconfiguration interface (`reconfig_pll1`)
- Transceiver reconfiguration interface (`xcvr_reconfig`)
- Hard IP reconfiguration interface (`hip_reconfig`)

If you later want to enable the Link Inspector using the same design, you need to provide a free-running clock and a reset to drive these interfaces at the top level of the PCIe IP. Intel recommends that you generate a new design example with the Link Inspector enabled. When you do so, the design example will include a free-running clock and a reset for all reconfiguration interfaces.

D.2.1.1. Enabling the PCIe Link Inspector

You enable the PCIe Link Inspector on the **Configuration Debug and Extension Options** tab of the parameter editor. You must also turn on the following parameters to use the **PCIe Link Inspector**:

- **Enable transceiver dynamic reconfiguration**
- **Enable dynamic reconfiguration of PCIe read-only registers**
- **Enable Native PHY, LCPLL, and fPLL ADME for Transceiver Toolkit**

To have access to the low-level link status information such as the information from the LTSSM, XCVR, and PLLs using the PCIe Link Inspector from the top level of the PCIe IP, you can enable the **Enable PCIe Link Inspector AVMM Interface** option. This allows you to extract the information from the `pli_avmm_*` ports for link-level debugging without JTAG access. This optional debug capability requires you to build custom logic to read and write data from the PCIe Link Inspector.

Note: The IP GUI only exposes the **Enable PCIe Link Inspector AVMM Interface** option if you enable the **Enable PCIe Link Inspector** option.

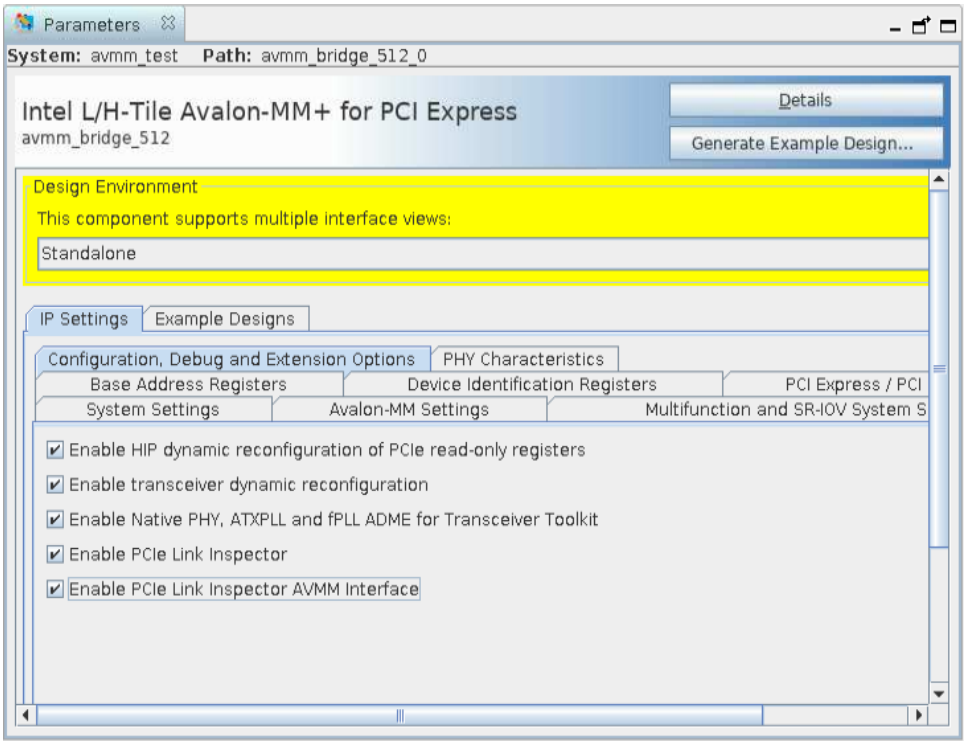
Table 71. PCIe Link Inspector Avalon-MM Interface Ports

Signal Name	Direction	Description
`pli_avmm_master_clk`	Input	Clock for the Avalon-MM defined interface
`pli_avmm_master_reset`	Input	Active-low Avalon-MM reset
`pli_avmm_master_write`	Input	Write signal
`pli_avmm_master_read`	Input	Read signal
`pli_avmm_master_address[19:0]`	Input	20-bit address
`pli_avmm_master_writedata[31:0]`	Input	32-bit write data

continued...

Signal Name	Direction	Description
`pli_avmm_master_waitrequest`	Output	When asserted, this signal indicates that the IP core is not ready to respond to a request.
`pli_avmm_master_readdatavalid`	Output	When asserted, this signal indicates that the data on `pli_avmm_master_readdata[31:0]` is valid.
`pli_avmm_master_readdata[31:0]`	Output	32-bit read data

Figure 37. Enable the Link Inspector in the Intel L-/H-Tile Avalon-MM+ for PCI Express IP



By default, all of these parameters are disabled.

For the design example generation, a JTAG-to-Avalon Bridge instantiation is connected to the exported `pli_avmm_*` ports, so that you can read all the link information via JTAG. The JTAG-to-Avalon Bridge instantiation is to verify the `pli_avmm_*` ports through JTAG. Without the design example generation, the JTAG-to-Avalon Bridge instantiation is not present.

D.2.1.2. Launching the PCIe Link Inspector

Use the design example you compiled in the *Quick Start Guide*, to familiarize yourself with the PCIe Link Inspector. Follow the steps in the *Generating the Avalon-ST Design* or *Generating the Avalon-MM Design* and *Compiling the Design* to generate the SRAM Object File, (`.sof`) for this design example.

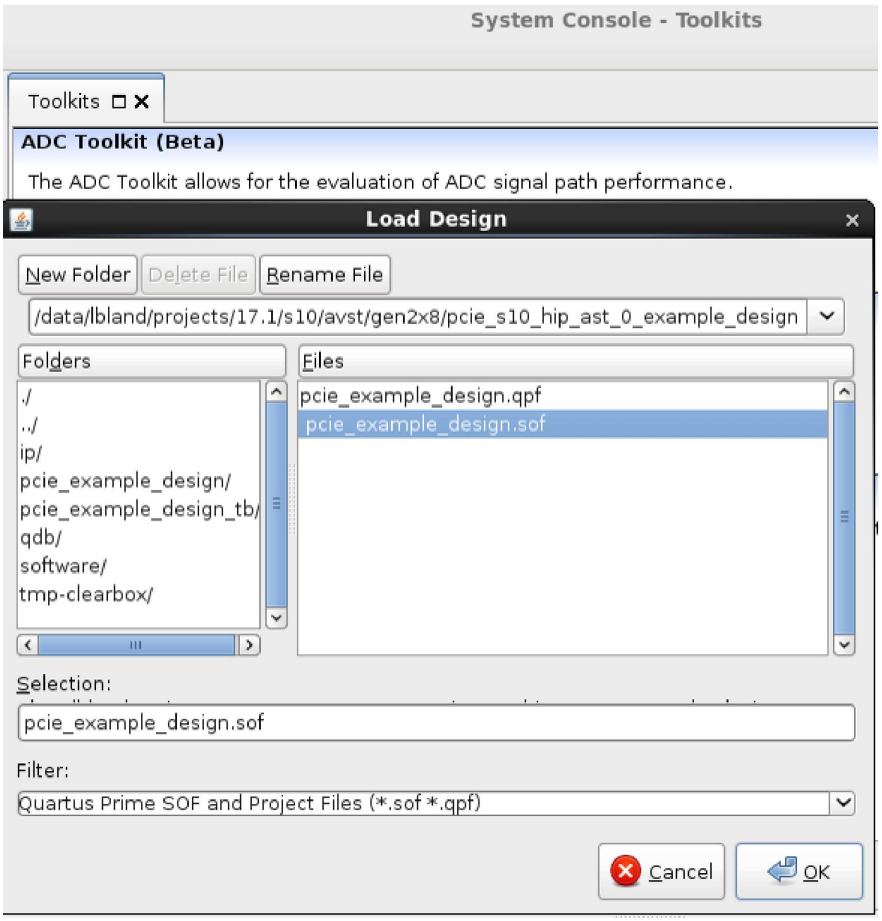
To use the PCIe Link Inspector, download the `.sof` to the Stratix 10 Development Kit. Then, open the System Console on the test PC and load the design to the System Console as well. Loading the `.sof` to the System Console allows the System Console to communicate with the design using NPDME. NPDME is a JTAG-based Avalon-MM master. It drives an Avalon-MM slave interfaces in the PCIe design. When using NPDME, the Quartus Prime software inserts the debug interconnect fabric to connect with JTAG.

Here are the steps to complete these tasks:

1. Use the Quartus Prime Programmer to download the `.sof` to the Stratix 10 FPGA Development Kit.

   Note: To ensure that you have the correct operation, you must use the same version of the Quartus Prime Programmer and Quartus Prime Pro Edition software that you used to generate the `.sof`.

2. To load the design to the System Console:
   a. Launch the Quartus Prime Pro Edition software on the test PC.
   b. Start the System Console, **Tools ➤ System Debugging Tools ➤ System Console**.
   c. On the System Console File menu, select **Load design** and browse to the `.sof` file.



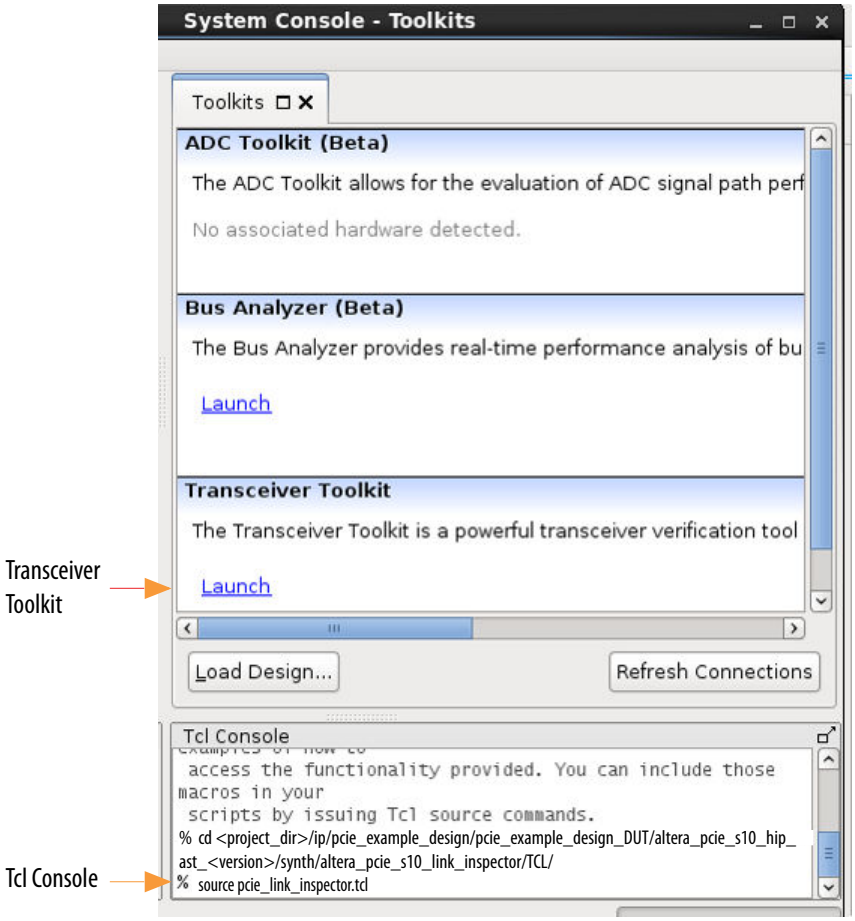
d. Select the `.sof` and click **OK**.
The `.sof` loads to the System Console.

3. In the System Console Tcl console, type the following commands:

```
% cd  <project_dir>/ip/pcie_example_design/
pcie_example_design_DUT/altera_pcie_s10_hip_ast_<version>/synth/
altera_pcie_s10_link_inspector
% source TCL/setup_adme.tcl
% source TCL/xcvr_pll_test_suite.tcl
% source TCL/pcie_link_inspector.tcl
```

The `source TCL/pcie_link_inspector.tcl` command automatically outputs the current status of the PCIe link to the Tcl Console. The command also loads all the PCIe Link Inspector functionality.

Figure 38. Using the Tcl Console to Access the PCIe Link Inspector



D.2.1.3. The PCIe Link Inspector LTSSM Monitor

The LTSSM monitor stores up to 1024 LTSSM states and additional status information in a FIFO. When the FIFO is full, it stops storing. Reading the LTSSM offset at address 0x02 empties the FIFO. The `ltssm_state_monitor.tcl` script implements the LTSSM monitor commands.

D.2.1.3.1. LTSSM Monitor Registers

You can program the LTSSM monitor registers to change the default behavior.

Table 72. LTSSM Registers

Base Address	LTSSM Address	Access	Description
0x20000 (5)	0x00	RW	`LTSSM Monitor Control` register. The `LTSSM Monitor Control` includes the following fields: • [1:0]: `Timer Resolution Control`. Specifies the number of `hip_reconfig_clk` the PCIe link remains in each LTSSM state. The following encodings are defined: — 2'b00: The main timer increments each `hip_reconfig_clk` cycle. This is the default value. — 2'b01: The main timer increments each 16 `hip_reconfig_clk` cycles. — 2'b10: The main timer increments each 256 `hip_reconfig_clk` cycles. — 2'b11: The main timer increments each <*n*> `hip_reconfig_clk` cycles. The `Timer Resolution Step` field defines <*n*>. • [17:2]: `Timer Resolution Step`. Specifies the value of <*n*> when `Timer Resolution Control` = 2'b11. • [18]: `LTSSM FIFO reset`. The following encodings are defined: — 1'b0: The LTSSM FIFO operates normally. — 1'b1: The LTSSM FIFO is in reset. • [19]: Reserved. • [20]: `LTSSM State Match Enable`. The following encodings are defined: — 1'b0: The LTSSM State match function is disabled — 1'b1: The LTSSM State match function is enabled.When the current LTSSM state matches a state stored in the `LTSSM State Match` register, the `State Match Flag` asserts. • [27:22] `LTSSM State Match`. When enabled, the LTSSM monitor compares the value in this register against each LTSSM state. If the values match, the LTSSM state match flag (offset address 0x01, bit 29) is set to 1. • [31:28]: Reserved.
	0x01	RO	`LTSSM Quick Debug Status` register. The `LTSSM Quick Debug Status` register includes the following fields: • [9:0]: `Number LTSSM States`. Specifies the number of states currently stored in the FIFO. • [10]: `LTSSM FIFO Full Flag`. When asserted, the LTSSM FIFO is full. • [11]: `LTSSM FIFO Empty Flag`. When asserted, the LTSSM FIFO is empty. • [12]: `Current PERSTN Status`. Stores the current PERSTN value. • [13]: `Current SERDES PLL Locked`. The following encodings are defined: — 1'b0: The SERDES PLL is not locked. — 1'b1: The SERDES PLL is locked. • [14]: `PCIe Link Status`. The following encodings are defined: — 1'b0: The link is down. — 1'b1: The link is up.

continued...

(5) When the **Enable PCIe Link Inspector AVMM Interface** option is **On**, the base address of the LTSSM Registers becomes 0x8000. Use this value to access these registers via the `pli_avmm_master_address[19:0]` ports.

Base Address	LTSSM Address	Access	Description
			• [16:15] `Current PCIe Data Rate`. The following encodings are defined: — 2'b00: Reserved. — 2'b01=Gen1. — 2'b10=Gen2. — 2'b11=Gen3. • [17]: `Native PHY Channel Locked to Data`. The following encodings are defined: — 1'b0: At least one CDR channel is not locked to data. — 1'b1: All CDR channels are locked to data. • [21:18]: `Current Number of PCIe Active Lanes`. • [22]: Reserved. • [28:23]: `Current LTSSM State`. • [29]: `LTSSM State Match Flag`. Asserts when the current state matches the state you specify in `LTSSM State Match`. • [31:30]: Reserved.

continued...

Base Address	LTSSM Address	Access	Description
	0x02	RO	LTSSM FIFO Output. Reading this register is equivalent to reading one entry from the LTSSM FIFO. Reading this register also updates the LTSSM FIFO, 0x03. The following fields are defined: • [5:0] LTSSM State. • [7:6]: PCIe Current Speed. • [12:8:] PCIe Lane Act. • [13]: SerDes PLL Locked. • [14]: Link Up. • [15]: PERSTN. • [16]:Native PHY Channel 0. When asserted, the CDR is locked to data. • [17]: Native PHY Channel 1. When asserted, the CDR is locked to data. • [18]: Native PHY Channel 2. When asserted, the CDR is locked to data. • [19]: Native PHY Channel 3. When asserted, the CDR is locked to data. • [20]: Native PHY Channel 4. When asserted, the CDR is locked to data. • [21]: Native PHY Channel 5. When asserted, the CDR is locked to data. • [22]: Native PHY Channel 6. When asserted, the CDR is locked to data. • [23]: Native PHY Channel 7. When asserted, the CDR is locked to data. • [24]: Native PHY Channel 8. When asserted, the CDR is locked to data. • [25]: Native PHY Channel 9. When asserted, the CDR is locked to data. • [26]: Native PHY Channel 10. When asserted, the CDR is locked to data. • [27]: Native PHY Channel 11. When asserted, the CDR is locked to data. • [29]: Native PHY Channel 12. When asserted, the CDR is locked to data. • [28]: Native PHY Channel 13. When asserted, the CDR is locked to data. • [30]: Native PHY Channel 14. When asserted, the CDR is locked to data. • [31]: Native PHY Channel 15. When asserted, the CDR is locked to data.
	0x03	RO	LTSSM FIFO Output [63:32] [29:0] Main Timer. The timer resets to 0 on each LTSSM transition. The value in this register indicates how long the PCIe link remains in each LTSSM state.
	0x04	RW	LTSSM Skip State Storage Control register. Use this register to specify a maximum of 4 LTSSM states. When LTSSM State Skip Enable is on, the LTSSM FIFO does not store the specified state or states. Refer to LTSSM State Encodings for the LTSSM Skip Field for the state encodings. [5:0]: LTSSM State 1. [6]: LTSSM State 1 Skip Enable. [12:7]: LTSSM State 2. [13]: LTSSM State 2 Skip Enable. [19:14]: LTSSM State 3. [20]: LTSSM State 3 Skip Enable. [26:21]: LTSSM State 4. [27]: LTSSM State 4 Skip Enable.

Table 73. LTSSM State Encodings for the LTSSM Skip Field

State	Encoding
Detect.Quiet	6'h00
Detect.Active	6'h01
Polling.Active	6'h02
Polling.Compliance	6'h03

continued...

State	Encoding
Polling.Configuration	6'h04
PreDetect.Quiet	6'h05
Detect.Wait	6'h06
Configuration.Linkwidth.Start	6'h07
Configuration.Linkwidth.Accept	6'h08
Configuration.Lanenum.Wait	6'h09
Configuration.Lanenum.Accept	6'h0A
Configuration.Complete	6'h0B
Configuration.Idle	6'h0C
Recovery.RcvrLock	6'h0D
Recovery.Speed	6'h0E
Recovery.RcvrCfg	6'h0F
Recovery.Idle	6'h10
Recovery.Equalization Phase 0	6'h20
Recovery.Equalization Phase 1	6'h21
Recovery.Equalization Phase 2	6'h22
Recovery.Equalization Phase 3	6'h23
L0	6'h11
L0s	6'h12
L123.SendEIdle	6'h13
L1.Idle	6'h14
L2.Idle	6'h15
L2.TransmitWake	6'h16
Disabled.Entry	6'h17
Disabled.Idle	6'h18
Disabled	6'h19
Loopback.Entry	6'h1A
Loopback.Active	6'h1B
Loopback.Exit	6'h1C
Loopback.Exit.Timeout	6'h1D
HotReset.Entry	6'h1E
Hot.Reset	6'h1F

D.2.1.3.2. ltssm_save2file <file_name>

The `ltssm_save2file <file_name>` command empties and LTSSM FIFO and saves all states to the file name you specify.

```
% ltssm_save2file <file_name>
```

D.2.1.3.3. ltssm_file2console

You can use the command `ltssm_file2console` to display the contents of a previously saved file on the TCL system console window. For example, if you previously used the command `ltssm_save2file` to save all the LTSSM states and other status information into a file, you can use `ltssm_file2console` to display the saved contents of that file on the TCL system console window.

This is a new command that is added in the 18.1 release of Quartus Prime.

D.2.1.3.4. ltssm_debug_check

The `ltssm_debug_check` command returns the current PCIe link status. It provides general information about the health of the link. Because this command returns real-time data, an unstable link returns different states every time you run it.

```
% ltssm_debug_check
```

Sample output:

```
################################################################
#####          LTSSM Quick Debugs Check Status          #####
################################################################
This PCIe is GEN1X8
Pin_Perstn signal is High
SerDes PLL is Locked
Link is Up
NOT all Channel CDRs are Locked to Data
LTSSM FIFO is Full
LTSSM FIFO is NOT Empty
LTSSM FIFO stored 1024 data
Current LTSSM State is 010001 L0
```

D.2.1.3.5. ltssm_display <num_states>

The `ltssm_display <num_states>` command reads data from the LTSSM FIFO and outputs <num_states> LTSSM states and associated status to the System Console.

```
% ltssm_display <num_states>
```

Sample output:

LTSSM[4:0]	Perstn	Locktodata	Link Up	Active Lanes	Current Speed	Timer	PLL Locked
0x00 Detect.Quiet	0	0x0000	0	0	00	0	0
0x00 Detect.Quiet	1	0x00FF	1	8	00	183237	1
0x11 L0	1	0x00FF	1	8	01	26607988	1
0x0D Recovery.Rcvlock	1	0x00FF	1	8	01	252	1
0x0F Recovery.Rcvconfig	1	0x00FF	1	8	01	786	1
0x0E Recovery.Speed	1	0x00FF	1	8	02	175242	1
0x0D Recovery.Rcvlock	1	0x00FF	1	8	02	6291458	1
0x0E Recovery.Speed	1	0x00FF	1	8	01	175296	1

continued...

0x0D Recovery.Rcvlock	1	0x00FF	1	8	01	2628	1
0x0F Recovery.Rcvconfig	1	0x00FF	1	8	0001	602	1
0x00 Recovery.Idle	1	0x00FF			0001	36	1

You can use this command to empty the FIFO by setting <num_states> to 1024:

```
% ltssm_display 1024
```

D.2.1.3.6. ltssm_save_oldstates

If you use `ltssm_display` or `ltssm_save2file` to read the LTSSM states and other status information, all that information is cleared after the execution of these commands. However, you can use the `ltssm_save_oldstates` command to overcome this hardware FIFO read clear limitation.

The first time you use `ltssm_save_oldstates`, it saves all states to the file provided at the command line. If you use this command again, it appends new readings to the same file and displays on the TCL console all the LTSSM states saved to the file.

This is a new command that is added in the 18.1 release of Quartus Prime.

D.2.1.4. Accessing the Configuration Space and Transceiver Registers

D.2.1.4.1. PCIe Link Inspector Commands

These commands use the PCIe Link Inspector connection to read and write registers in the Configuration Space, LTSSM monitor, PLLs, and Native PHY channels.

Table 74. PCIe Link Inspector (PLI) Commands

These commands are available in the `link_insp_test_suite.tcl` script.

Command	Description
`pli_read32 <slave_if> <pli_base_addr> <pli_reg_addr>`	Performs a 32-bit read from the slave interface at the base address and register address specified.
`pli_read8 <slave_if> <base_addr> <reg_addr>`	Performs an 8-bit read from the slave interface at the base address and register address specified.
`pli_write32 <slave_if> <pli_base_addr> <pli_reg_addr> <value>`	Performs a 32-bit write of the value specified to the slave interface at the base address and the register address specified.
`pli_write8 <slave_if> <base_addr> <reg_addr> <value>`	Performs an 8-bit write of the value specified to the slave interface at the base address and the register address specified.
`pli_rmw32 <slave_if> <base_addr> <reg_addr> <bit_mask> <value>`	Performs a 32-bit read-modify-write of the value specified to the slave interface to the slave interface at the base address and register address using the bit mask specified.
`pli_rmw8 <slave_if> <base_addr> <reg_addr> <bit_mask> <value>`	Performs an 8-bit read-modify-write to the slave interface at the base address and register address using the bit mask specified.
`pli_dump_to_file <slave_if> <filename> <base_addr> <start_reg_addr> <end_reg_addr>`	Writes the contents of the slave interface to the file specified. The base address and the start and end register addresses specify range of the write.

continued...

Command	Description
	The <slave_if> argument can have the following values: • $atxpll • $fpll • $channel(<n>)

PCIe Link Inspector Command Examples

The following commands use the addresses specified below in the *Register Address Map*.

Use the following command to read register 0x480 from the ATX PLL:

```
% pli_read8 $pli_adme $atxpll_base_addr 0x480
```

Use the following command to write 0xFF to the fPLL register at address 0x4E0:

```
% pli_write8 $pli_adme $fpll_base_addr 0x4E0 0xFF
```

Use the following command to perform a read-modify-write to write 0x02 to channel 3 with a bit mask of 0x03 for the write:

```
% pli_rmw8 $pli_adme $xcvr_ch3_base_addr 0x481 0x03 0x02
```

Use the following command to instruct the LTSSM monitor to skip recording of the Recovery.Rcvlock state:

```
$pli_write $pli_adme $ltssm_base_addr 0x04 0x0000000D
```

D.2.1.4.2. NPDME PLL and Channel Commands

These commands use the Native PHY and channel PLL NPDME master ports to read and write registers in the ATX PLL, fPLL, and Native PHY transceiver channels.

Table 75. NPDME Commands To Access PLLs and Channels

These commands are available in the `xcvr_pll_test_suite.tcl`

Command	Description
`adme_read32 <slave_if> <reg_addr>`	Performs a 32-bit read from the slave interface of the register address specified.
`adme_read8 <slave_if> <reg_addr>`	Performs an 8-bit read from the slave interface of the register address specified.
`adme_write32 <slave_if> <reg_addr> <value>`	Performs a 32-bit write of the value specified to the slave interface and register specified
`adme_write8 <slave_if> <reg_addr> <value>`	Performs an 8-bit write of the value specified to the slave interface and register specified
`adme_rmw32 <slave_if> <reg_addr> <bit_mask> <value>`	Performs a 32-bit read-modify-write to the slave interface at the register address using the bit mask specified.
`adme_rmw8 <slave_if> <reg_addr> <bit_mask> <value>`	Performs an 8-bit read-modify-write to the slave interface at the register address using the bit mask specified.

continued...

Command	Description
`adme_dump_to_file <slave_if> <filename> <start_addr> <end_addr>`	Writes the contents of the slave interface to the file specified. The start and end register addresses specify the range of the write. The `<slave_if>` argument can have the following values: • `$atxpll` • `$fpll` • `$channel(<n>)`
`atxpll_check`	Checks the ATX PLL lock and calibration status
`fpll_check`	Checks the fPLL lock and calibration status
`channel_check`	Checks each channel clock data recovery (CDR) lock status and the TX and RX calibration status

NPDME Command Examples

The following PLL commands use the addresses specified below in the *Register Address Map*.

Use the following command to read the value register address 0x480 in the ATX PLL:

```
% adme_read8 $atxpll_adme 0x480
```

Use the following command to write 0xFF to register address 0x4E0 in the fPLL:

```
% adme_write8 $fpll_adme 0x4E0 0xFF
```

Use the following command to perform a read-modify-write to write register address 0x02 in channel 3:

```
% adme_rmw8 $channel_adme(3) 0x03 0x02
```

Use the following command to save the register values from 0x100-0x200 from the ATX PLL to a file:

```
% adme_dump_to_file $atxpll <directory_path>atx_regs.txt 0x100 0x200
```

D.2.1.4.3. Register Address Map

Here are the base addresses when you run the as defined in `link_insp_test_suite.tcl` and `ltssm_state_monitor.tcl`.

Table 76. PCIe Link Inspector and LTSSM Monitor Register Addresses

Base Address	Functional Block	Access
0x00000	fPLL	RW
0x10000	ATX PLL	RW
0x20000	LTSSM Monitor	RW
0x40000	Native PHY Channel 0	RW
0x42000	Native PHY Channel 1	RW

continued...

Base Address	Functional Block	Access
0x44000	Native PHY Channel 2	RW
0x46000	Native PHY Channel 3	RW
0x48000	Native PHY Channel 4	RW
0x4A000	Native PHY Channel 5	RW
0x4C000	Native PHY Channel 6	RW
0x4E000	Native PHY Channel7	RW
0x50000	Native PHY Channel 8	RW
0x52000	Native PHY Channel 9	RW
0x54000	Native PHY Channel 10	RW
0x56000	Native PHY Channel 11	RW
0x58000	Native PHY Channel 12	RW
0x5A000	Native PHY Channel 13	RW
0x5C000	Native PHY Channel 14	
0x5E000	Native PHY Channel 15	RW
0x80000	PCIe Configuration Space	RW

Related Information

Logical View of the L-Tile/H-Tile Transceiver Registers
For detailed register descriptions for the ATX PLL, fPLL, PCS, and PMA registers.

D.2.1.5. Additional Status Commands

D.2.1.5.1. Displaying PLL Lock and Calibration Status Registers

1. Run the following System Console Tcl commands to display the lock and the calibration status for the PLLs and channels.

   ```
   % source TCL/setup_adme.tcl
   ```

   ```
   % source TCL/xcvr_pll_suite.tcl
   ```

2. Here are sample transcripts:

```
##########################################
#####         ATXPLL  Status         #####
##########################################
ATXPLL is Locked
ATXPLL Calibration is Done
##########################################
#####          FPLL  Status          #####
##########################################
FPLL is Locked
FPLL Calibration is Done
##########################################
#####       Channel# 0 Status        #####
##########################################
Channel#0 CDR is Locked to Data
Channel#0 CDR is Locked to Reference Clock
Channel#0 TX Calibration is Done
Channel#0 RX Calibration is Done
##########################################
...
##########################################
```

```
Channel#7 CDR is Locked to Data
Channel#7 CDR is Locked to Reference Clock
Channel#7 TX Calibration is Done
Channel#7 RX Calibration is Done
```

E. Root Port Enumeration

This chapter provides a flow chart that explains the Root Port enumeration process.

The goal of enumeration is to find all connected devices in the system and for each connected device, set the necessary registers and make address range assignments.

At the end of the enumeration process, the Root Port (RP) must set the following registers:

- Primary Bus, Secondary Bus and Subordinate Bus numbers
- Memory Base and Limit
- IO Base and IO Limit
- Max Payload Size
- Memory Space Enable bit

The Endpoint (EP) must also have the following registers set by the RP:

- Master Enable bit
- BAR Address
- Max Payload Size
- Memory Space Enable bit
- Severity bit

The figure below shows an example tree of connected devices on which the following flow chart will be based.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Figure 39. Tree of Connected Devices in Example System

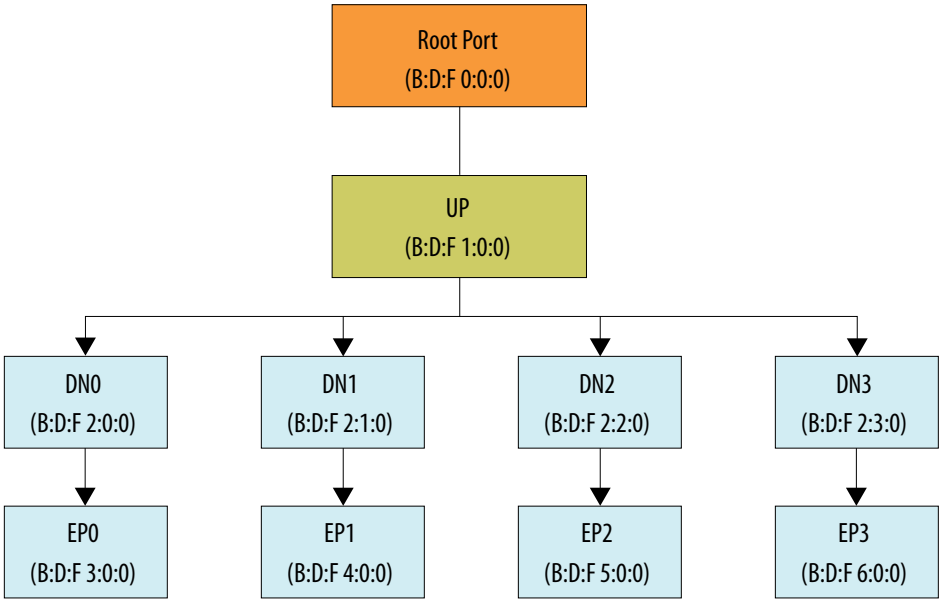


Figure 40. Root Port Enumeration Flow Chart

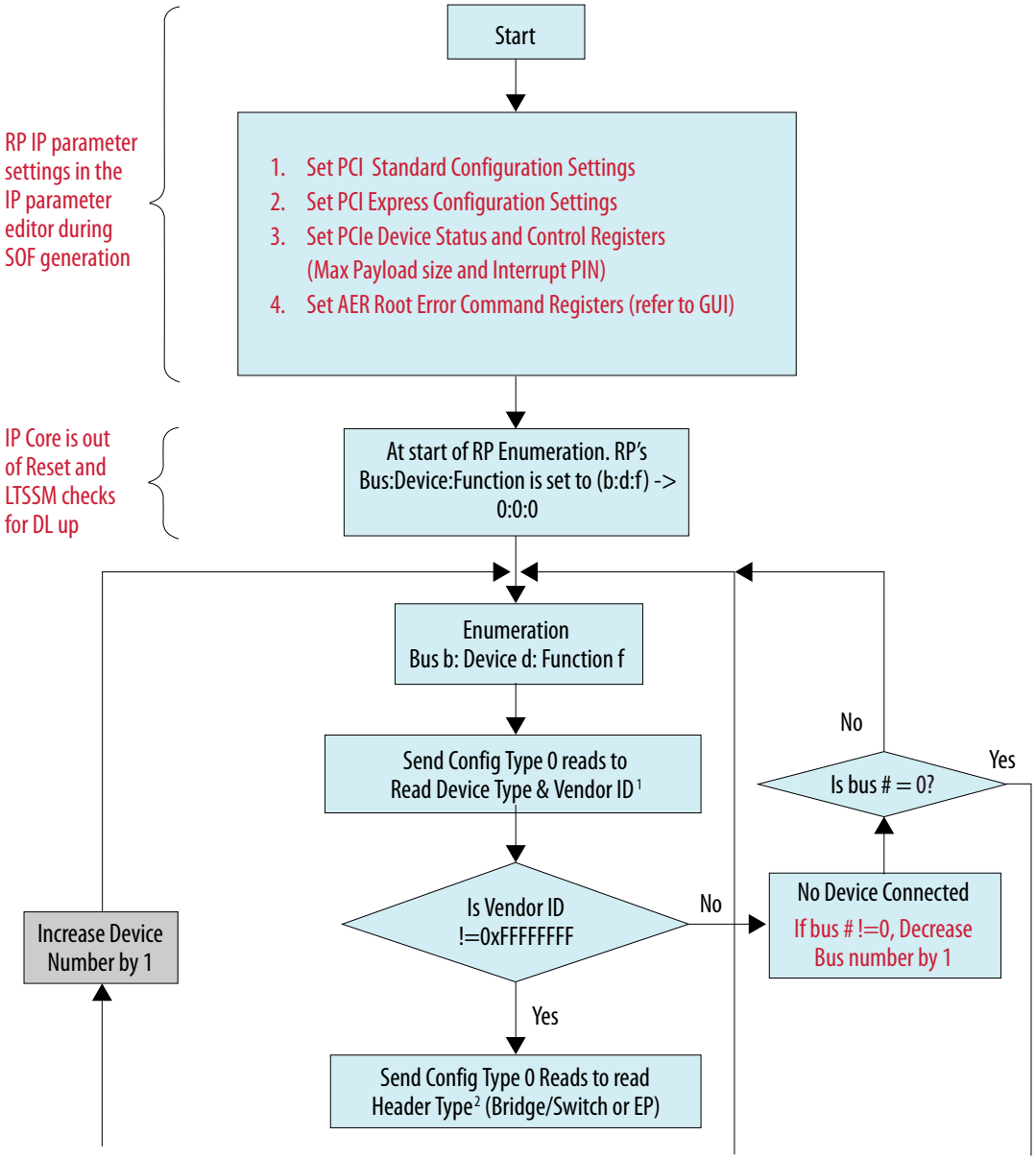


Figure 41. Root Port Enumeration Flow Chart (continued)

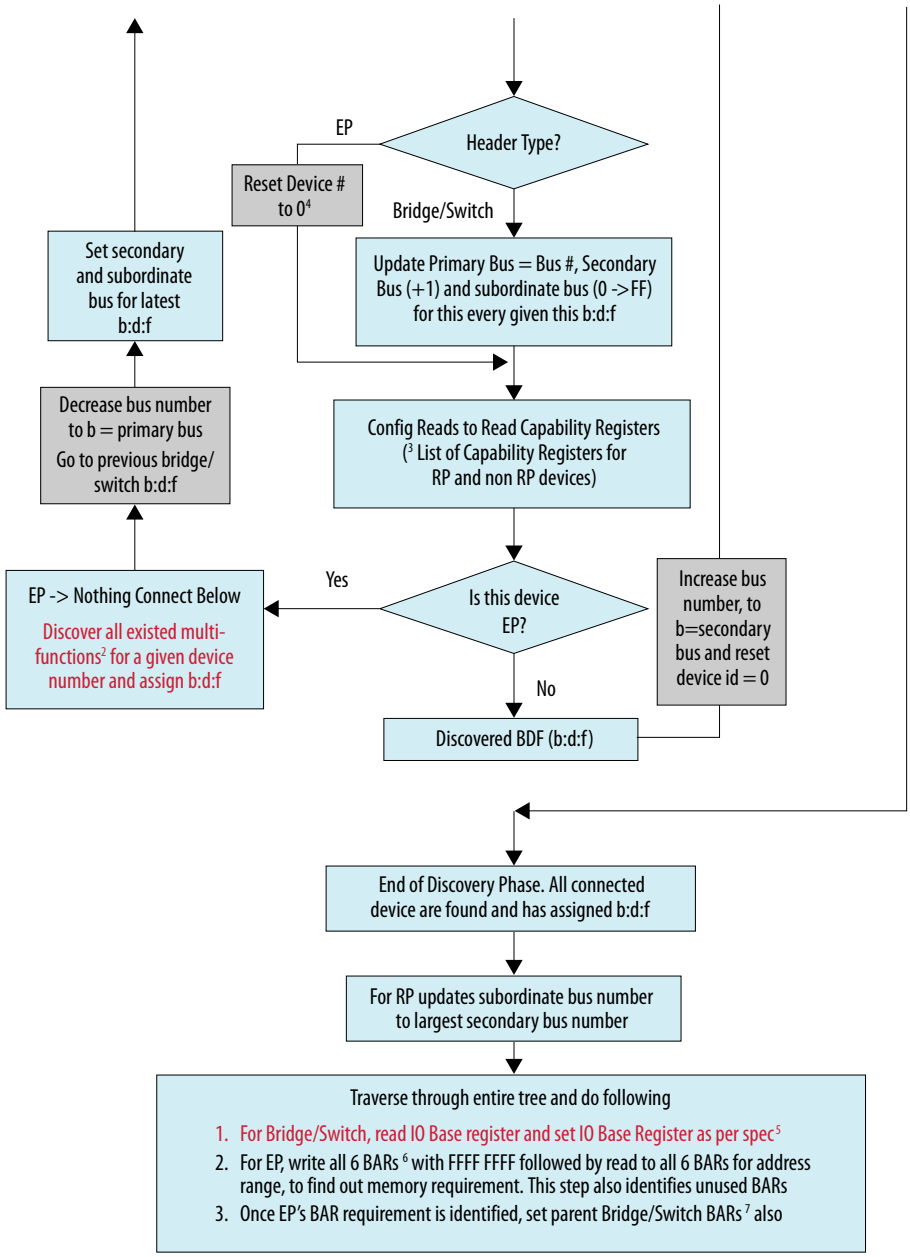
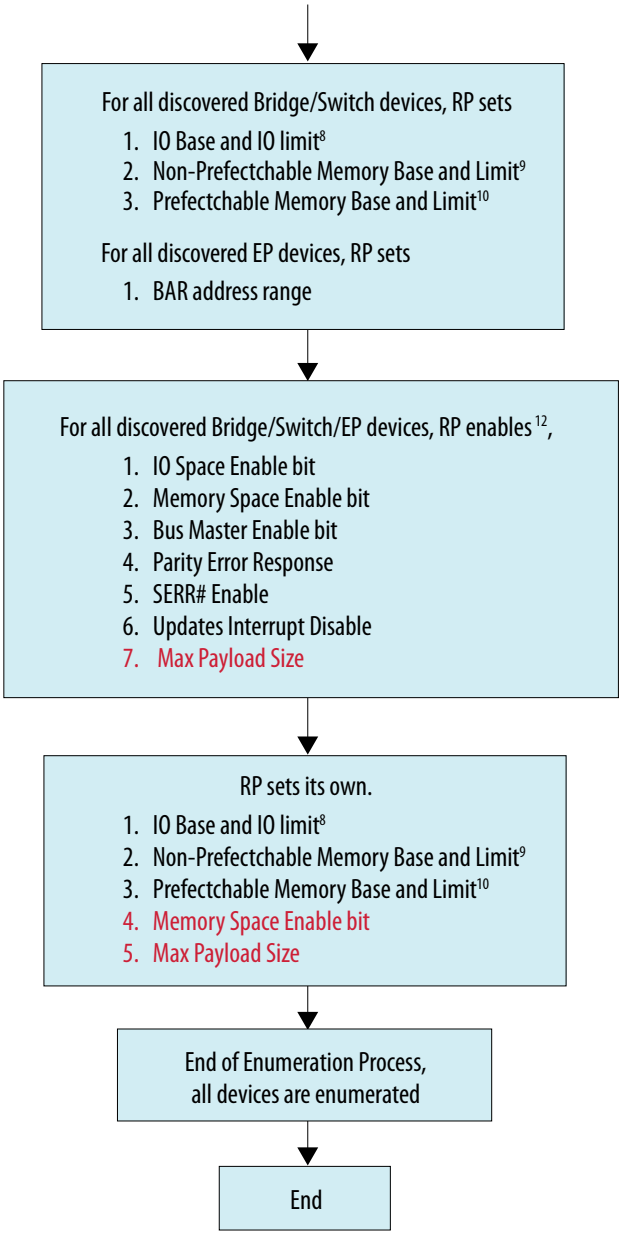


Figure 42. Root Port Enumeration Flow Chart (continued)



Notes:

1. Vendor ID and Device ID information is located at offset 0x00h for both Header Type 0 and Header Type 1.
2. For PCIe Gen4, the Header Type is located at offset 0x0Eh (2nd DW). If bit 0 is set to 1, it indicates the device is a Bridge; otherwise, it is an EP. If bit 7 is set to 0, it indicates this is a single-function device; otherwise, it is a multi-function device.
3. List of capability registers for RP and non-RP devices:

- 0x34h – Capabilities Pointers. This register is used to point to a linked list of capabilities implemented by a Function:
  a. Capabilities Pointer for RP
     i. Address 40 - Identifies the Power Management Capability ID
     ii. Address 50 - Identifies MSI Capability ID
     iii. Address 70 - Identifies the PCI Express Capability structure
  b. Capabilities Pointer for non-RP
     i. Address 40 - Identifies Power Management Capability ID
     ii. Address 70 - Identifies the PCI Express Capability structure

4. EP does not have an associated register of Primary, Secondary and Subordinate Bus numbers.
5. Bridge/Switch IO Base and Limit register offset 0x1Ch. These registers are set per the PCIe 4.0 Base Specification. For more accurate information and flow, refer to chapter 7.5.1.3.6 of the Base Specification.
6. For EP Type 0 header, BAR addresses are located at the following offsets:
   a. 0x10h – Base Address 0
   b. 0x14h – Base Address 1
   c. 0x18h – Base Address 2
   d. 0x1ch – Base Address 3
   e. 0x20h – Base Address 4
   f. 0x24h – Base Address 5
7. For Bridge/Switch Type 1 header, BAR addresses are located at the following offsets:
   a. 0x10h – Base Address 0
   b. 0x14h – Base Address 1
8. For Bridge/Switch Type 1 header, IO Base and IO limit registers are located at offset 0x1Ch.
9. For Bridge/Switch Type 1 header, Non-Prefetchable Memory Base and Limit registers are located at offset 0x20h.
10. For Bridge/Switch Type 1 header, Prefetchable Memory Base and Limit registers are located at offset 0x24h.
11. For Bridge/Switch/EP Type 0 & 1 headers, the Bus Master Enable bit is located at offset 0x04h (Command Register) bit 2.
12. For Bridge/Switch/EP Type 0 & 1 headers,
    a. IO Space Enable bit is located at offset 0x04h (Command Register) bit 0.
    b. Memory Space Enable bit is located at offset 0x04h (Command Register) bit 1.
    c. Bus Master Enable bit is located at offset 0x04h (Command Register) bit 2.
    d. Parity Error Response bit is located at offset 0x04h (Command Register) bit 6.
    e. SERR# Enable bit is located at offset 0x04h (Command Register) bit 8.
    f. Interrupt Disable bit is located at offset 0x04h (Command Register) bit 10.