Intel FPGA P-Tile Avalon Memory Mapped (Avalon-MM) IP for PCI Express Design Example User Guide

Updated for Intel® Quartus® Prime Design Suite: 19.4
IP Version: 1.1.0
Contents

1. Design Example Overview........................................................................................................3
   1.1. Block Descriptions........................................................................................................4
       1.1.1. DMA Controller................................................................................................4
       1.1.2. Avalon-MM Address to PCIe Address Mapping................................................7
       1.1.3. BAR Interpreter................................................................................................8
   1.2. Programming Model for the Design Example.................................................................8
   1.3. Descriptor Format for the Design Example....................................................................9
   1.4. DMA Operations Using the Design Example...............................................................10
       1.4.1. Read DMA Example........................................................................................10
       1.4.2. Write DMA Example....................................................................................10

2. Quick Start Guide................................................................................................................12
   2.1. Design Components.....................................................................................................12
   2.2. Directory Structure....................................................................................................13
   2.3. Generating the Design Example..................................................................................13
   2.4. Compiling the Design Example..................................................................................15
   2.5. Installing the Linux Kernel Driver.............................................................................16
   2.6. Running the Design Example Application..................................................................17

1. Design Example Overview

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.

The example design also connects the Bursting Master (in non-bursting mode) 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).

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 BAR Interpreter also connects the Bursting Master to the dual-port memory.

The example design is generated dynamically based on the selected variation of the P-Tile Avalon®-MM IP for PCIe. 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.

In the 19.4 release of Intel® Quartus® Prime, the only variation supported is the DMA variation. This variation instantiates the Bursting Master (in non-bursting mode), Read Data Mover and Write Data Mover. 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.
Note: Beginning with the 17.1 release, the Intel Quartus Prime Pro Edition software dynamically generates example designs for the parameters you specify in the parameter editor. Consequently, the Intel Quartus Prime Pro Edition installation directory no longer provides static example designs for Intel Stratix® 10 devices. Static example designs are available for earlier device families, including Intel Arria® 10 and Intel Cyclone® 10 devices.

1.1. Block Descriptions

The DMA design example for the P-Tile Avalon-MM IP for PCIe includes the following components:

- **DUT**: The P-Tile Avalon-MM IP for PCIe Endpoint.
- **MEM0**: An on-chip dual-port memory that connects to the Read Data Mover and Write Data Mover interfaces of the DUT.
- **DMA_CONTROLLER**: A DMA Controller that interfaces with the normal and priority descriptor queues of the DUT’s Read Data Mover and Write Data Mover.
- **BAR_INTERPRETER**: A BAR Interpreter that combines the address and BAR number to form a wider address that Platform Designer can use to route memory transactions to the various slaves. The BAR Interpreter connects the Bursting Master of the DUT to the dual-port memory.

**Figure 2. Platform Designer View of the DMA Design Example for the P-Tile Avalon-MM IP for PCIe**

1.1.1. DMA Controller

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 P-Tile Avalon-MM 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.

### 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 1. Register Set of the DMA Controller**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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.
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.

### 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 backpressure 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.
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>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Interrupt always</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Interrupt if error</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>No interrupt</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>No interrupt and drop status word (i.e, do not even write it to the WS or RS status queues)</td>
</tr>
</tbody>
</table>

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.

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.

1.1.2. 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.

1.1.3. 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.

1.2. Programming Model for the Design Example

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 174-bit long, 512 bits are reserved for each descriptor. The descriptors are LSB-aligned in that 512-bit field.

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.
1.3. Descriptor Format for the Design Example

The Read and Write Data Movers use descriptors to transfer data. The descriptor format is fixed and specified below:

Table 3. Descriptor Format for Data Movers

<table>
<thead>
<tr>
<th>Signals Description (for <code>rddm_desc_data_i</code> or <code>wrdm_desc_data_i</code>)</th>
<th>Read Data Mover</th>
<th>Write Data Mover</th>
</tr>
</thead>
<tbody>
<tr>
<td>[173:160]: reserved</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>[159:152]: descriptor ID</td>
<td>ID of the descriptor</td>
<td>ID of the descriptor</td>
</tr>
<tr>
<td>[151:149]: application-specific</td>
<td>Application-specific bits. Example of an Intel application is provided below.</td>
<td>Application-specific bits. Example of an Intel application is provided below.</td>
</tr>
<tr>
<td>[148]: single destination</td>
<td>When the single destination bit is set, the same destination address is used for all the transfers. If the bit is not set, the address increments for each transfer.</td>
<td>N/A</td>
</tr>
<tr>
<td>[147]: single source</td>
<td>N/A</td>
<td>When the single source bit is set, the same source address is used for all the transfers. If the bit is not set, the address increments for each transfer. Note that in single source mode, the PCIe address and Avalon-MM address must be 64-byte aligned.</td>
</tr>
<tr>
<td>[146]: immediate</td>
<td>N/A</td>
<td>When set, the immediate bit indicates immediate writes. Immediate writes of one or two dwords are supported. For immediate transfers, bits [31:0] or [63:0] contain the payload for one- or two-dword transfers respectively. The two-dword immediate writes cannot cross a 4k boundary. This can be used for MSI/MSI-X for example.</td>
</tr>
<tr>
<td>[145:128]: transfer size</td>
<td>Number of dwords to transfer (up to 1 MB).</td>
<td>Number of dwords to transfer (up to 1 MB).</td>
</tr>
<tr>
<td>[127:64]: destination address</td>
<td>Avalon-MM address</td>
<td>PCIe Address</td>
</tr>
<tr>
<td>[63:0]: source address</td>
<td>PCIe Address</td>
<td>Avalon-MM address</td>
</tr>
</tbody>
</table>

Application-Specific Bits

Three application-specific bits (bits [151:149]) from the Write Data Mover and Read Data Mover Status Avalon-ST Source interfaces control when interrupts are generated.

Table 4. Encodings for Application-Specific Bits

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Interrupt always</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Interrupt if error</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>No interrupt</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No interrupt and drop status word</td>
</tr>
</tbody>
</table>
The External 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 the 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.

1.4. DMA Operations Using the Design Example

1.4.1. 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 \text{rddm\_desc\_ready\_o} or \text{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.

1.4.2. 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.
2. Quick Start Guide

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

2.1. Design Components

The available design example is for an Endpoint with a single function. This DMA design example includes a DMA Controller and an on-chip memory to exercise the Data Movers in the P-Tile Avalon-MM IP for PCI Express.

Figure 3. Block Diagram for the Platform Designer Avalon-MM with DMA Design Example

---

*Other names and brands may be claimed as the property of others.*
2.2. Directory Structure

Figure 4. Directory Structure for the Generated Design Example

2.3. Generating the Design Example

Figure 5. Design Example Generation Procedure

1. In the Intel 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 **Intel Agilex** or **Intel Stratix 10**.

6. If you select **Intel Stratix 10** in the last step, select **Stratix 10 DX** in the **Device** pull-down menu.

7. Select the **Target Device** for your design.

8. Click **Finish**.

9. In the IP Catalog locate and add the **Intel P-Tile Avalon-MM IP for PCI Express**.

10. In the **New IP Variant** dialog box, specify a name for your IP. Click **Create**.

11. On the **Top-Level Settings** and **PCIe** Settings tabs, specify the parameters for your IP variation.

12. On the **Example Designs** tab, make the following selections:
   
   a. 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.
   
   b. For **Generated HDL Format**, only Verilog is available in the current release.
   
   c. For **Target Development Kit**, select the appropriate option.

   Note: For the current release of the IP core, hardware testing is not supported. Therefore, select **None** for the target development kit. The generated design example will target the device you specified in Step 5 above.

13. Select **Generate Example Design** to create a design example that you can simulate. When the prompt asks you to specify the directory for your example design, you can accept the default directory, `<example_design>/intel_pcie_ptile_avmm_0_example_design`, or choose another directory.

**Figure 6. Example Designs Tab**

14. Click **Finish**. You may save your .ip file when prompted, but it is not required to be able to use the example design.
2.4. Compiling the Design Example

1. Navigate to `<project_dir>/intel_pcie_ptile_avmm_0_example_design/` and open `pcie_ed.qpf`.

2. If you are using the Intel Stratix 10 DX development kit, add the following assignments in the `.qsf` file of your project for the VID feature:
   - `set_global_assignment -name USE_CONF_DONE SDM_IO16`
   - `set_global_assignment -name VID_OPERATION_MODE "PMBUS MASTER"`
   - `set_global_assignment -name USE_PWRMGT_SCL SDM_IO0`
   - `set_global_assignment -name USE_PWRMGT_SDA SDM_IO12`
   - `set_global_assignment -name PWRMGT_BUS_SPEED_MODE "100 KHZ"`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE_TYPE OTHER`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE0_ADDRESS 60`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE1_ADDRESS 00`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE2_ADDRESS 00`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE3_ADDRESS 00`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE4_ADDRESS 00`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE5_ADDRESS 00`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE6_ADDRESS 00`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE7_ADDRESS 00`
   - `set_global_assignment -name PWRMGT_VOLTAGE_OUTPUT_FORMAT "DIRECT FORMAT"`
   - `set_global_assignment -name PWRMGT_DIRECT_FORMAT_COEFFICIENT_M 1`
   - `set_global_assignment -name PWRMGT_DIRECT_FORMAT_COEFFICIENT_R 0`
   - `set_global_assignment -name PWRMGT_TRANSLATED_VOLTAGE_VALUE_UNIT MILLIVOLTS`
   - `set_global_assignment -name PWRMGT_PAGE_COMMAND_ENABLE ON`

3. If you are using the Intel Agilex development kit, add the following assignments in the `.qsf` file of your project for the VID feature:
   - `set_global_assignment -name USE_CONF_DONE SDM_IO16`
   - `set_global_assignment -name USE_INIT_DONE SDM_IO0`
   - `set_global_assignment -name USE_CVP_CONF_DONE SDM_IO15`
   - `set_global_assignment -name VID_OPERATION_MODE "PMBUS MASTER"`
   - `set_global_assignment -name USE_PWRMGT_SCL SDM_IO14`
   - `set_global_assignment -name USE_PWRMGT_SDA SDM_IO11`
   - `set_global_assignment -name PWRMGT_BUS_SPEED_MODE "100 KHZ"`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE_TYPE OTHER`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE0_ADDRESS 47`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE1_ADDRESS 00`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE2_ADDRESS 00`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE3_ADDRESS 00`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE4_ADDRESS 00`
   - `set_global_assignment -name PWRMGT_SLAVE_DEVICE5_ADDRESS 00`
2.5. 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\(^1\) reads and writes
- Configuration Space DWORD reads and writes

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 that 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 1172:0000 -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.

---

\(^1\) Throughout this user guide, the terms word, DWORD and QWORD have the same meaning that they have in the PCI Express Base Specification. A word is 16 bits, a DWORD is 32 bits, and a QWORD is 64 bits.
Expected result:
Kernel driver in use: intel_fpga_pcieDrv

2.6. 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 Intel Stratix 10 DX or Intel Agilex 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 Intel Stratix 10 DX or Intel Agilex Development Kit, you can determine the BDF by typing the following command:
$ lspci -d 1172

4. Here are sample transcripts for automatic and manual modes:

<table>
<thead>
<tr>
<th>Intel FPGA PCIe Link Test - Automatic Mode</th>
<th>Intel FPGA PCIe Link Test - Manual Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 2.0</td>
<td>Version 1.0</td>
</tr>
<tr>
<td>0: Automatically select a device</td>
<td>0: Automatically select a device</td>
</tr>
<tr>
<td>1: Manually select a device</td>
<td>1: Manually select a device</td>
</tr>
<tr>
<td>********************************************</td>
<td>******************************************</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>Opened a handle to BAR 0 of a device with BDF 0x100</td>
<td>Enter bus number:</td>
</tr>
<tr>
<td>********************************************</td>
<td>Enter device number:</td>
</tr>
<tr>
<td>0: Link test - 100 writes and reads</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>1: Write memory space</td>
<td>Doing 100 writes and 100 reads . .</td>
</tr>
<tr>
<td>2: Read memory space</td>
<td>Number of write errors: 0</td>
</tr>
<tr>
<td>3: Write configuration space</td>
<td>Number of read errors: 0</td>
</tr>
<tr>
<td>4: Read configuration space</td>
<td>Number of DWORD mismatches: 0</td>
</tr>
<tr>
<td>5: Change BAR</td>
<td>******************************************</td>
</tr>
<tr>
<td>6: Change device</td>
<td>******************************************</td>
</tr>
<tr>
<td>7: Enable SR-IOV</td>
<td>******************************************</td>
</tr>
<tr>
<td>8: Do a link test for every enabled virtual function belonging to the current device</td>
<td>******************************************</td>
</tr>
<tr>
<td>9: Perform DMA</td>
<td>******************************************</td>
</tr>
<tr>
<td>10: Quit program</td>
<td>******************************************</td>
</tr>
<tr>
<td>********************************************</td>
<td>******************************************</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>Number of write errors: 0</td>
<td>Number of device number:</td>
</tr>
<tr>
<td>Number of read errors: 0</td>
<td>******************************************</td>
</tr>
</tbody>
</table>
Enter function number:
> 0

BDF is 0x100

Enter BAR number (-1 for none):
> 4

Opened a handle to BAR 4 of a device with BDF 0x100
A. Document Revision History for the Intel FPGA P-Tile Avalon Memory Mapped (Avalon-MM) Design Example for PCI Express User Guide

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>IP Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.12.16</td>
<td>19.4</td>
<td>1.1.0</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>