PCI Express DMA Reference Design Using External Memory

Updated for Intel® Quartus® Prime Design Suite: **Quartus Prime Pro v17.0 Arria 10**

Online Version

Send Feedback

ID: **683390**

AN-708

Version: **2017.05.17**

Contents

Send Feedback

683390 | 2017.05.17

Send Feedback

PCI Express DMA Reference Design Using External Memory

The PCI Express® DMA reference design using external memory highlights the performance of the Intel® Arria® V, Arria 10, Cyclone® V and Stratix® V Hard IP for PCI Express using the Avalon® Memory-Mapped (Avalon-MM) interface. The design includes a high-performance DMA with an Avalon-MM interface that connects to the PCI Express Hard IP and a DDR3 (for Cyclone V, Arria V and Stratix V devices) or DDR4 (for Arria 10 devices) memory controller. It transfers data between an external memory and host system memory. The reference design includes a Linux and Windows based software driver that sets up the DMA transfer. You can also use the software driver to measure and display the performance achieved for the transfers. This reference design allows you to evaluate the performance of the Hard IP for PCI Express using the Avalon-MM interface.

Related Information

- V-Series Avalon-MM DMA Interface for PCIe Solutions User Guide
- Arria 10 Hard IP for PCI Express IP Cores
- PCI Express Base Specification Revision 3.0
- Arria V Reference Design
  To download the reference design and the design software
- Arria 10 Reference Design
  To download the reference design and the design software
- Cyclone V Reference Design
  To download the reference design and the design software
- Stratix V Reference Design
  To download the reference design and the design software.

Deliverables Included with the Reference Design

The reference design includes the following components:

- Linux and Windows applications and drivers configured specifically for this reference design. Arria V and Cyclone V support Linux applications and drivers. Arria 10 and Stratix V support Linux as well as Windows applications and drivers.
- FPGA programming files for the Arria V, Arria 10, Cyclone V, or Stratix V FPGA Development kits
- Quartus® Prime Archive Files (**. qar**) for the development kits, including an SRAM Object File (**. sof**) and SignalTap® files (**.stp**)
- An Application Layer which is an Avalon-MM DMA example design generated in Qsys

Project Hierarchy

The reference design uses the following directory structures:

- The top-level module is **top_hw**.
- `top` — includes all design files. If you modify the design, you must copy the modified files into this folder before recompiling the design.

Parameter Settings for PCI Express Hard IP Variation

The Hard IP for PCI Express variant used in this reference design supports a 256-byte maximum payload size. The following tables list the values for all parameters.

Table 1. System Settings

Parameter	Value
Number of lanes	Arria V, Cyclone V: x4 Arria 10, Stratix V: x8
Lane rate	Arria V, Cyclone V: Gen2 (5.0 Gbps) Arria 10, Stratix V: Gen3 (8.0 Gbps)
RX buffer credit allocation – performance for received request	Low
Reference clock frequency	100 MHz
Enable configuration via the PCIe link	Disabled
Use ATX PLL	Disabled

Table 2. Base Address Register (BAR) Settings

Parameter	Value
BAR0 Type	64-bit prefetchable memory
BAR0 Size	512 Bytes – 9 bits
BAR1	Disabled
BAR2 Type	Arria V, Cyclone V: 64-bit prefetchable memory Arria 10Stratix V: Disabled
BAR2 Size	Arria V, Cyclone V: 64 MByte (MB) – 26 bits Arria 10, Stratix V: Disabled
BAR3	Disabled
BAR4 Type	Arria V, Cyclone V: Disabled Arria 10, Stratix V: 64-bit prefetchable memory
BAR4 Size	Arria V, Cyclone V: Disabled Arria 10, Stratix V: 256 MB – 28 bits
BAR5	Disabled

Table 3. Device Identification Register Settings

Parameter	Value
Vendor ID	0x00001172
Device ID	0x0000E003

continued...

Parameter	Value
Revision ID	0x00000001
Class Code	Arria V, Cyclone V, Stratix V: 0x00000000 Arria 10: 0x00ff0000
Subsystem Vendor ID	Arria V, Cyclone V, Stratix V: 0x00000000 Arria 10: 0x00004e1b
Subsystem Device ID	Arria V: 0x00000101 Cyclone V: 0x00000000 Arria 10: 0x00004e0c Stratix V: 0x0000b862

Table 4. PCI Express/PCI Capabilities

Parameter	Value
Maximum payload size	Cyclone V: 128 Bytes Arria V, Arria 10, Stratix V: 256 Bytes
Completion timeout range	ABCD
Implement Completion Timeout Disable	Enabled

Table 5. Error Reporting Settings

Parameter	Value
Advanced error reporting (AER)	Disabled
ECRC checking	Disabled
ECRC generation	Disabled

Table 6. Link Settings

Parameter	Value
Link port number	1
Slot clock configuration	Enabled

Table 7. MSI and MSI-X Settings

Parameter	Value
Number of MSI messages requested	Arria V, Arria 10, Cyclone V: 4 Stratix V: 1
Implement MSI-X	Disabled
Table size	0
Table offset	0x0000000000000000
Table BAR indicator	0
Pending bit array (PBA) offset	0x0000000000000000
PBA BAR Indicator	0

Table 8. Power Management

Parameter	Value
Endpoint L0s acceptable latency	Maximum of 64 ns
Endpoint L1 acceptable latency	Maximum of 1 us

Table 9. PCIe Address Space Setting

Parameter	Value
Address width of accessible PCIe memory space	Arria V, Cyclone V: 32 Arria 10: 40 Stratix V: 64

Quartus Prime Settings

The **.qar** file in the reference design package has the recommended synthesis, fitter, and timing analysis settings for the parameters specified in this reference design.

Avalon-MM DMA Controller

Read Data Mover

The Read Data Mover sends memory read TLPs upstream. After the completion is received, it writes the received data to the external DDR3 or DDR4 memory.

Write Data Mover

The Write Data Mover reads data from the external DDR3 or DDR4 memory and sends it upstream using memory write TLPs on the PCI Express link.

Descriptor Controller

The Descriptor Controller module manages the DMA read and write operations. This module is included in the IP Core to facilitate the customization of descriptor handling. Optionally, you can use an external descriptor controller for your application.

Inside the controller, a FIFO stores the descriptors which are fetched by the Read Data Mover. Host software programs the internal registers of the Descriptor Controller with the location and size of the descriptor table residing in the host system memory through the Avalon-MM RX master port. Based on this information, the descriptor controller directs the Read Data Mover to copy the entire table and store it in the internal FIFO. The controller then fetches the table entries and directs the DMA controller to transfer the data between the Avalon-MM and PCIe domains one descriptor at a time. It also sends DMA status upstream via the Avalon-MM TX slave (TXS) port.

TX Slave

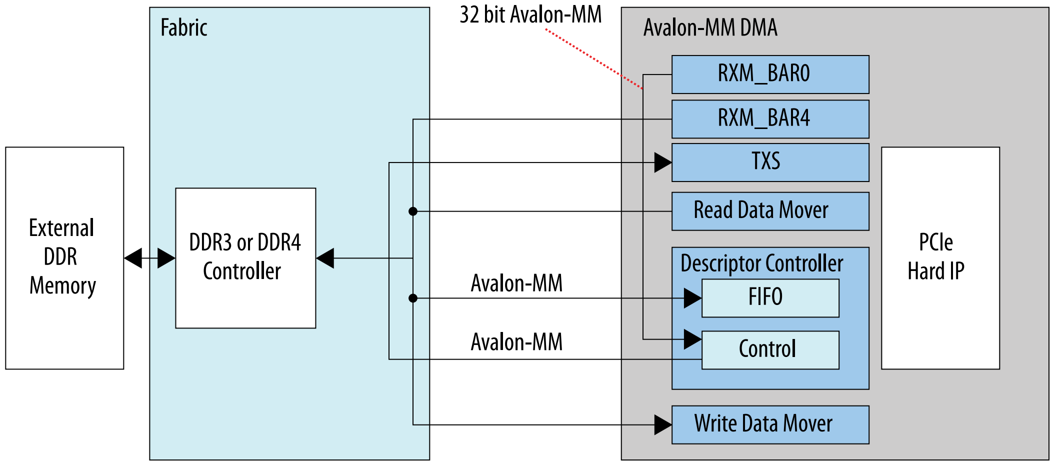
The TX Slave module propagates Avalon-MM reads and writes upstream. External Avalon-MM masters, including the DMA control master, can access system memory using the TX Slave. The DMA Controller uses this path to update the DMA status upstream, using Message Signaled Interrupt (MSI) TLPs.

RX Master

The RX Master module propagates single dword read and write TLPs from the Root Port to the Avalon-MM domain via a 32-bit Avalon-MM master port. Software instructs the RX Master to send control, status, and descriptor information to Avalon-MM slaves, including the DMA control slave.

Reference Design

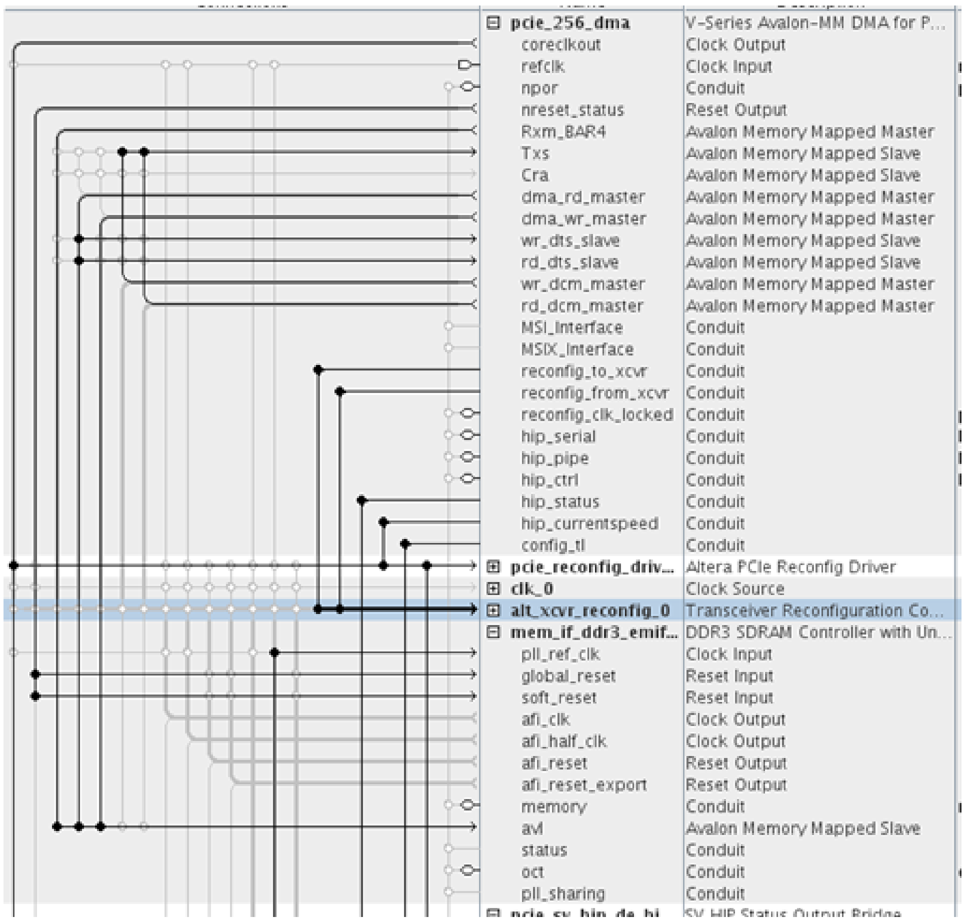
Figure 1. DMA Reference Design Block Diagram



The reference design uses an external memory connected to the Intel memory controller that can access up to 128 MB of on-board external memory. The read DMA moves the data from the system memory to the external memory. The write DMA moves the data from the external memory to the system memory.

Because the memory contains a single port, the read and the write data movers cannot access the external memory at the same time. Hence, this reference design does not demonstrate the real capability of the DMA for simultaneous reads and writes. Because the latency of the external memory is higher than that of an on-chip memory, the throughput achieved with an external memory is lower compared to the throughput of on-chip memory.

Figure 2. DMA Reference Design Qsys Connections



⊟ pcie_256_dma	V-Series Avalon-MM DMA for P...
coreclkout	Clock Output
refclk	Clock Input
npor	Conduit
nreset_status	Reset Output
Rxm_BAR4	Avalon Memory Mapped Master
Txs	Avalon Memory Mapped Slave
Cra	Avalon Memory Mapped Slave
dma_rd_master	Avalon Memory Mapped Master
dma_wr_master	Avalon Memory Mapped Master
wr_dts_slave	Avalon Memory Mapped Slave
rd_dts_slave	Avalon Memory Mapped Slave
wr_dcm_master	Avalon Memory Mapped Master
rd_dcm_master	Avalon Memory Mapped Master
MSI_Interface	Conduit
MSIX_Interface	Conduit
reconfig_to_xcvr	Conduit
reconfig_from_xcvr	Conduit
reconfig_clk_locked	Conduit
hip_serial	Conduit
hip_pipe	Conduit
hip_ctrl	Conduit
hip_status	Conduit
hip_currentspeed	Conduit
config_tl	Conduit
⊞ pcie_reconfig_driv...	Altera PCIe Reconfig Driver
⊞ clk_0	Clock Source
⊞ alt_xcvr_reconfig_0	Transceiver Reconfiguration Co...
⊟ mem_if_ddr3_emif...	DDR3 SDRAM Controller with Un...
pll_ref_clk	Clock Input
global_reset	Reset Input
soft_reset	Reset Input
afi_clk	Clock Output
afi_half_clk	Clock Output
afi_reset	Reset Output
afi_reset_export	Reset Output
memory	Conduit
avl	Avalon Memory Mapped Slave
status	Conduit
oct	Conduit
pll_sharing	Conduit

Table 10. AVMM DMA Reference Design Blocks and Port Descriptions

Function	Port	Description
AVMM DMA	`pcie_256_dma`	This is the 256-bit Avalon Memory Mapped module with DMA. It consists of one read and one write data mover, a RX master, a TX slave, and an internal descriptor controller. The descriptor controller can also be external to the DMA module.
RXM_BAR0	N/A	This is an Avalon-MM master port. It passes the memory access from PCIe host to PCIe BAR0. The host uses this port to program the descriptor controller. Because this reference design uses an internal descriptor controller, the port connection is not shown in Qsys. The connection is inside the `pcie_256_dma` module.
RXM_BAR4	Rxm_BAR4	This is an Avalon-MM master port. It passes the memory access from PCIe host to PCIe BAR4. In the reference design, it connects to one port of the external DDR3 or DDR4 memory. The PCIe host accesses the memory through PCIe BAR4 In a typical application, system software controls this port to initialize random data in the external memory. Software also reads the data back to verify correct operation.

continued...

Function	Port	Description
TXS	`Txs`	This is an Avalon-MM slave port. In a typical application, an Avalon-MM master controls this port to send memory reads or writes to the PCIe domain. The descriptor controller uses it to write DMA status back to descriptor space in the PCIe domain when the DMA completes its operation. The descriptor controller also uses this port to send upstream MSI interrupts.
Read Data Mover	`dma_rd_master`	This is an Avalon-MM master port. The Read Data Mover moves data from the PCIe domain to the external DDR3 or DDR4 memory during normal read DMA operation. The Read Data Mover also fetches the descriptors from the PCIe domain and writes them to the FIFO in the Descriptor Controller. There are two separate groups of descriptors, one for write DMA and another for read DMA. Because of these two separate groups, the `dma_rd_master` port is connected to two ports. It connects to `wr_dts_slave` for the write DMA descriptor FIFO and `rd_dts_slave` for the read DMA descriptor FIFO.
Write Data Mover	`dma_wr_master`	This is an Avalon-MM master port. The Write Data Mover reads data from the external DDR3 or DDR4 memory and then writes data to the PCIe domain. In this reference design, because the DDR3 or DDR4 controller has a single port, the Write Data Mover uses the same port as the Read Data Mover.
FIFO in Descriptor Controller	`wr_dts_slave` and `rd_dts_slave`	This is an Avalon-MM slave port for the FIFO in the Descriptor Controller. When the Read Data Mover fetches the descriptors from system memory, it writes the descriptors to the FIFO using this port. Because there are two separate groups of descriptors for read and write, there are two ports. For the write DMA, the FIFO address is from `0x8000_2000`—`0x8000_3FFF`. For the read DMA, the FIFO address is from `0x8000_0000`—`0x8000_1FFF`.
Control in Descriptor Controller	`wr_dcm_master` and `rd_dcm_master`	The control block in the Descriptor Controller has one transmit and one receive port, one for read DMA and another one for write DMA. The receive port is connected to the RXM_BAR0 and the transmit port is connected to the `Txs`. The receive path from the RXM_BAR0 connects internally. It is not shown in the *DMA Reference Design Qsys Connections* figure. For the transmit path, both read and write DMA ports connect to the `Txs` externally. These ports are visible in the *DMA Reference Design Qsys Connections* figure.
DDR3 or DDR4 Controller	`Altera DDR3 or DDR4 controller`	This is a single port DDR3 or DDR4 controller. It can access up to 128 MB external memory.

DMA Operation Flow

Software completes the following steps to specify and initiate a DMA operation:

1. Software allocates free memory space in the system memory to populate the descriptor table.
2. Software allocates free space in the system memory for the data to be moved to and from the system memory by the DMA.
3. Software writes all descriptors into the descriptor table in the system memory. The DMA supports up to 128 descriptors. Each descriptor has an ID, from descriptor ID 0 to descriptor ID 127. Each descriptor contains the `source address`,

`destination address`, and size of the data to be moved. The source address specifies the location of the data to be moved from by the DMA. The destination address specifies the location that the data is moved to by the DMA.

4. For the read DMA operation, the software initializes the system memory space with random data. The Read Data Mover moves this data from the system memory to the external memory. For the write DMA operation, the software initializes the external memory with random data. The Write Data Mover moves the data from the external memory to the system memory space.
5. Software programs the registers in the Descriptor Controller's control logic through endpoint BAR0. Programming specifies the base address of the descriptor table which stores the descriptors in the system memory and the base address of the FIFO which is going to store the descriptors in the FPGA fabric domain. For this reference design, the base address is 0x80000000 for read DMA descriptor FIFO and 0x80002000 for write DMA descriptor FIFO.
6. As the last step to start the DMA, software writes the ID of the last descriptor into the Descriptor Controller's control logic which triggers the Descriptor Controller to start the DMA. The DMA then starts fetching the descriptors from descriptor 0 to the last descriptor.
7. After DMA fetches the last descriptor and transfers the data associated with that descriptor, the Descriptor Controller writes `1'b1` to the Done bit in the descriptor table header corresponding to the last descriptor in the PCIe domain through the `Txs` path.
8. Software polls the Done bit in the descriptor table header corresponding to the last descriptor. The DMA operation completes when the Done bit is set and the performance is calculated. Once the DMA completes, the software compares the data in the system memory to the external memory. The test passes when there is no data mismatch.
9. For simultaneous operation, the software begins the read DMA operation followed by the write DMA operation. The simultaneous operation is complete when both the read and write DMA operations finish.

Running the Reference Design

The reference design has the following hardware and software requirements:

Hardware Requirements

- The FPGA Development Kit
  - Arria V GX FPGA Development Kit
  - Arria 10 GX FPGA Development Kit
  - Cyclone V GX FPGA Development Kit
  - Stratix V GX FPGA Development Kit
- A computer with a PCIe slot running Windows or either 32- or 64 bit Linux.This computer is referred as computer number 1.
- A second computer with the Quartus Prime software installed on a computer running 64-bit Windows. This computer downloads the FPGA programming file (**.sof**) to the FPGA on the development kit. This computer is referred as computer number 2.
- A USB cable or other Intel FPGA download cable.

Software Requirements

- The reference design software installed on computer number 1.
- The Quartus Prime software version 15.1 or later installed on computer number 2.

Installing the Linux Software

On your Linux computer: :

1. Create a directory and unzip all files to that directory.
2. In the directory that you created, login as root by typing `su`.
3. Type your root password.
4. Type `make` to compile the driver and the application.
5. Type `./install` to install the Linux driver.

You will get the following error message when you install the driver for the first time:

```
ERROR: Module altera_dma does not exist in /proc/modules.
```

```
rm: cannot remove '/dev/altera_dma': no such file or directory.
```

Installing the Windows Software

On your Windows computer:

1. Download the `Windows_for_AVMM_DMA_On_Chip_Mem.zip` files for the demo driver. Extract the compressed files.
2. First, copy the `Windows_for_AVMM_DMA_On_Chip_Mem` directory to computer #1. Then, plug the PCI Express card into your Windows computer
3. Follow the instructions in README`.docx` in the `Windows_for_AVMM_DMA_On_Chip_Mem` directory to install and run the software application.

Installing the Software

1. Create a directory and unzip the Linux driver/application archive file to that directory.
2. In the directory that you created, login as root by typing `su`.
3. Type your root password.
4. Type `make` to compile the driver and the application.
5. Type `./install` to install the Linux driver.

Note: You will get the following error messages when you install the driver for the first time:

```
ERROR: Module altera_dma does not exist in /proc/modules.
```

```
rm: cannot remove '/dev/altera_dma': no such file or directory.
```

You can ignore these messages.

Installing the Hardware

1. Power down computer number 1.
2. Plug the FPGA Development kit into a PCI Express slot, being certain that the slot supports Gen3 x8 or Gen2 x4.

   The development kit has an integrated Intel FPGA Download Cable for FPGA programming.
3. Connect a USB cable from computer number 2 to the FPGA Development kit.
4. Turn on computer number 1.
5. On computer number 2, bring up the Quartus Prime programmer and program the FPGA through an Intel FPGA Download Cable.
6. To force system enumeration to discover the PCI Express IP core, reboot computer number 1.

Running the Linux DMA Software

1. In the terminal window of computer 1, change to the directory in which you installed the Linux driver.
2. Type `su`.
3. Type your super user password.
4. Type `make` to compile the driver and application.
5. Type `./install` to install the driver.
6. To run the DMA application, type `./run` .
   The application prints out the commands available to specify the DMA traffic you want to run. By default, the software enables DMA reads, DMA writes, and Simultaneous DMA reads and writes.

Table 11. Available Commands for DMA Operation

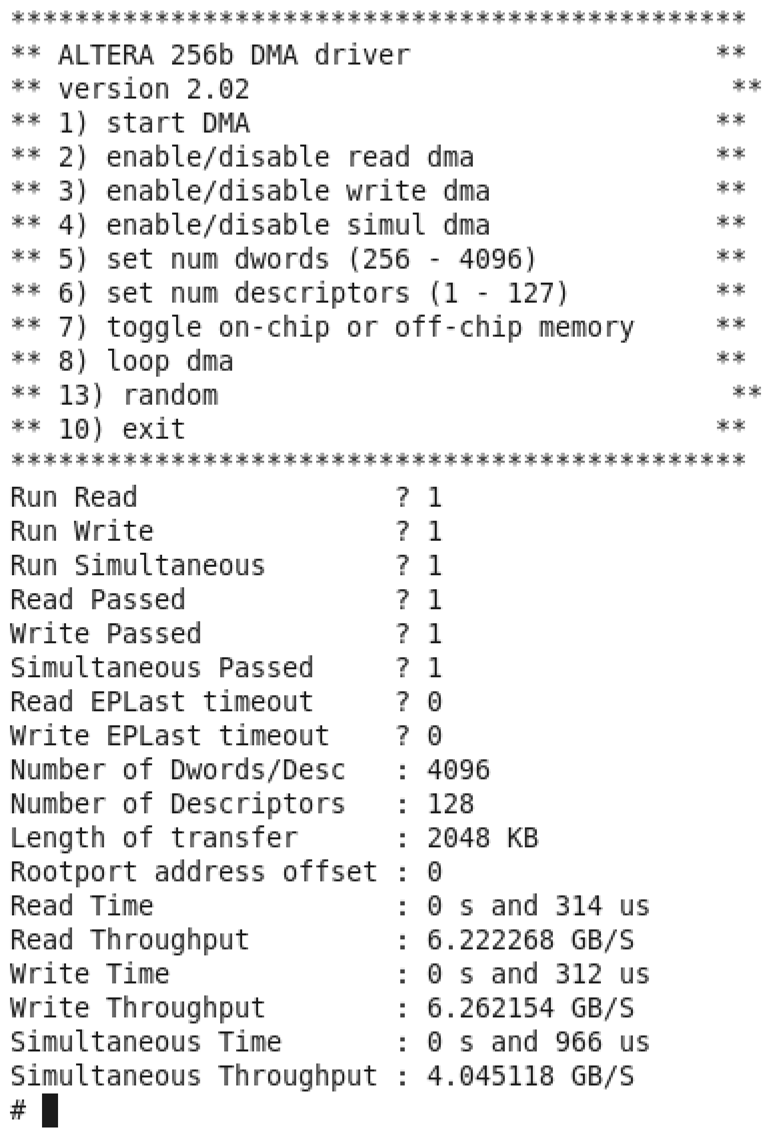
Command	Operation
1	Start the DMA
2	Enable or disable read DMA
3	Enable or disable write DMA
4	Enable or disable simultaneous read and write DMA
5	Set the number of dwords per descriptor. The legal range is 256-4096 dwords
6	Set the number of descriptors. The legal range is 1-127 descriptors.
8	Run a loop DMA.
10	Exit.

7. To start the DMA, type 1. This command runs the DMA for one loop. Type 8 to run the DMA in a loop.

The following figure shows the performance for DMA reads, DMA writes, and simultaneous DMA reads and writes.

Send Feedback

Figure 3. Output from 256-Bit DMA Driver



```
**********************************************
** ALTERA 256b DMA driver                   **
** version 2.02                              **
** 1) start DMA                             **
** 2) enable/disable read dma               **
** 3) enable/disable write dma              **
** 4) enable/disable simul dma              **
** 5) set num dwords (256 - 4096)           **
** 6) set num descriptors (1 - 127)         **
** 7) toggle on-chip or off-chip memory     **
** 8) loop dma                              **
** 13) random                                **
** 10) exit                                 **
**********************************************
Run Read                 ? 1
Run Write                ? 1
Run Simultaneous         ? 1
Read Passed              ? 1
Write Passed             ? 1
Simultaneous Passed      ? 1
Read EPLast timeout      ? 0
Write EPLast timeout     ? 0
Number of Dwords/Desc    : 4096
Number of Descriptors    : 128
Length of transfer       : 2048 KB
Rootport address offset : 0
Read Time                : 0 s and 314 us
Read Throughput          : 6.222268 GB/S
Write Time               : 0 s and 312 us
Write Throughput         : 6.262154 GB/S
Simultaneous Time        : 0 s and 966 us
Simultaneous Throughput : 4.045118 GB/S
# █
```

This DMA performance was achieved using the following PCIe Gen3 x8 in the system:

- Motherboard: Asus PBZ77-V
- Memory: Corsair 8 GB 2 dimm x [4GB (2x2GB)] @ 1600 MHz
- CPU:
  - Vendor_ID: GenuineIntel
  - CPU family: 6
  - Model: 58
  - Model name: Intel(R) Core(TM) i5-3450 CPU @ 3.10GHz
  - Stepping: 9
  - Microcode: 0xc
  - CPU MHz: 3101.000
  - Cache size: 6144 KByte (KB)

Running the Windows DMA Software

The Windows software package includes a document with instructions on how to run the application. The software application will be located in the Release_090_80000000 folder.

Understanding PCI Express Throughput

The throughput in a PCI Express system depends on the following factors:

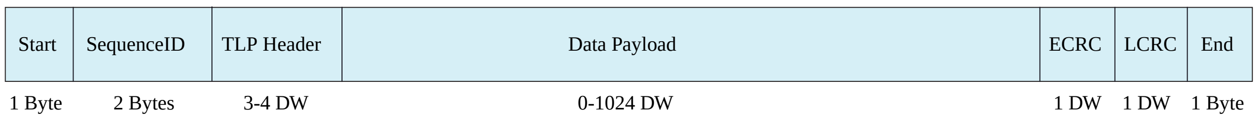
- Protocol overhead
- Payload size
- Completion latency
- Flow control update latency
- Devices forming the link

Protocol Overhead

Protocol overhead includes the following three components:

- 128b/130b Encoding and Decoding—Gen3 links use 128b/130b encoding. This encoding adds two synchronization (sync) bits to each 128-bit data transfer. Consequently, the encoding and decoding overhead is very small at 1.56%. The effective data rate of a Gen3 x8 link is about 8 Giga Byte per Second (GBps).
- Data Link Layer Packets (DLLPs) and Physical Layer Packets (PLPs)—An active link also transmits DLLPs and PLPs. The PLPs consist of SKP ordered sets which are 16-24 bytes. The DLLPs are two dwords. The DLLPs implement flow control and the ACK/NAK protocol.
- TLP Packet Overhead—The overhead associated with a single TLP ranges from 5-7 dwords if the optional ECRC is not included. The overhead includes the following fields:
  - — The Start and End Framing Symbols
  - — The Sequence ID
  - — A 3- or 4-dword TLP header
  - — The Link Cyclic Redundancy Check (LCRC)
  - — 0-1024 dwords of data payload

Figure 4. TLP Packet Format



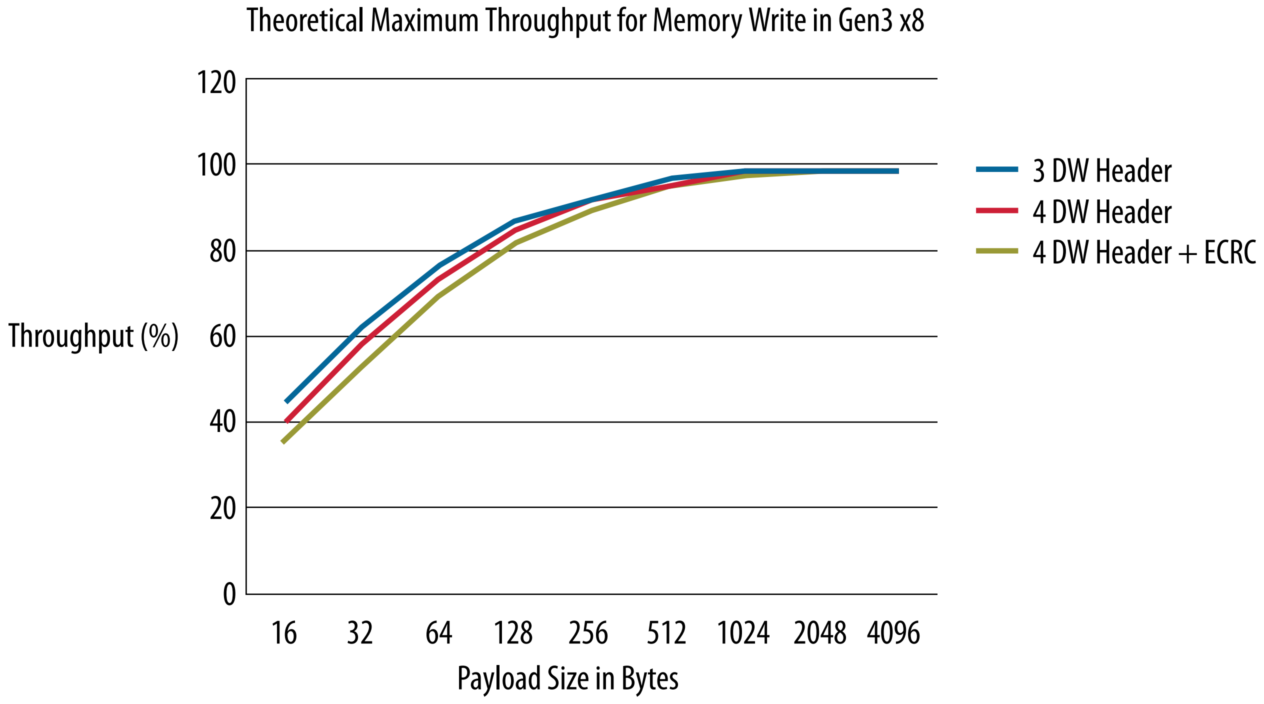
Throughput for Posted Writes

The theoretical maximum throughput is calculated using the following formula:

```
Throughput  = payload size / (payload size + overhead)
```

Figure 5. Maximum Throughput for Memory Writes

The graph shows the maximum throughput with different TLP header and payload sizes. The DLLPs and PLPs are excluded from this calculation. Based on the graph, the theoretical maximum throughput on a Gen3 x8 link for a 3-dword posted write and no ECRC is 89.8%. The maximum throughput is 8 GBps x 89.8% = 7.19 GBps.



Specifying the Maximum Payload Size

The `Device Control` register bits [7:5], specifies the maximum TLP payload size of the current system. The `Maximum Payload Size` field of the `Device Capabilities` register bits [2:0], specifies the maximum permissible value for the payload of the Hard IP for PCI Express IP Core. You specify this read-only parameter, called **Maximum Payload Size**, in the Hard IP for PCI Express Parameter Editor window. After determining the maximum TLP payload for the current system, software records that value in the `Device Control` register. This value must be less than the maximum payload specified in the `Maximum Payload Size` field of the `Device Capabilities` register.

Understanding Flow Control for PCI Express

Flow control guarantees that a TLP is not transmitted unless the receiver has enough buffer space to accept it. There are separate credits for headers and payload data. A device needs sufficient header and payload credits before sending a TLP. When the Application Layer in the completer accepts the TLP, it frees up the RX buffer space in the completer's Transaction Layer. The completer sends a flow control update packet (FC Update DLLP) to replenish the consumed credits to the initiator. If a device has used all its credits, transfers must stop until its credits are replenished. The throughput is limited by the rate at which the header and payload credits are replenished by sending FC Update DLLPs. The flow control updates depend on the maximum payload size and the latencies of the two connected devices.

Throughput for Reads

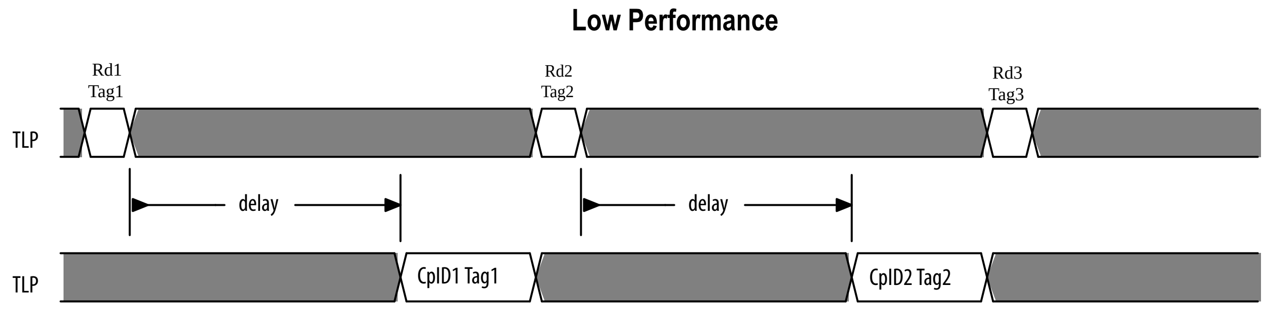
PCI Express uses a split transaction model for reads. The read transaction includes the following steps:

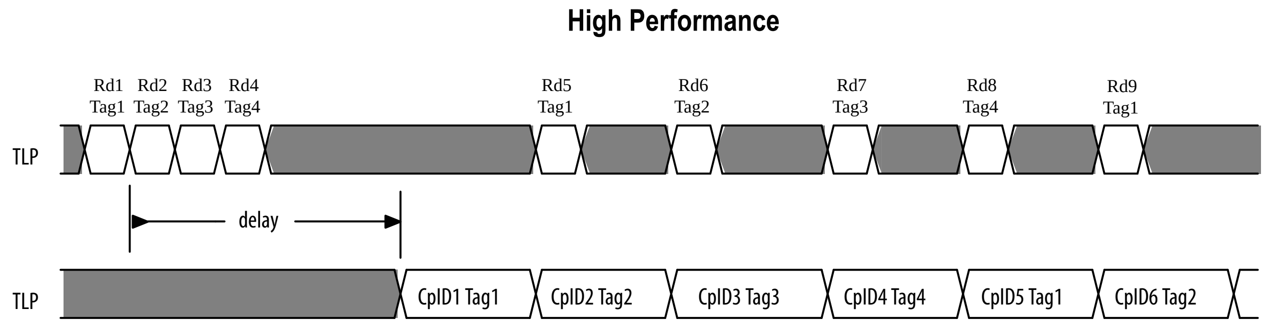
1. The requester sends a Memory Read Request.
2. The completer sends out the ACK DLLP to acknowledge the Memory Read Request.
3. The completer returns a Completion with Data. The completer can split the Completion into multiple completion packets.

Read throughput is typically lower than write throughput because reads require two transactions instead of a single write for the same amount of data. The read throughput depends on the delay between the time when the Application Layer issues a Memory Read Request and the time the completer takes to return data. To maximize the throughput, the application must issue enough outstanding read requests to cover this delay.

Figure 6. Read Request Timing

The figures below show the timing for Memory Read Requests (MRd) and Completions with Data (CplD). The first figure shows the requester waiting for the completion before issuing the subsequent requests. It results in lower throughput. The second figure shows the requester making multiple outstanding read requests to eliminate the delay after the first data returns. It has higher throughput.





To maintain maximum throughput for the completion data packets, the requester must optimize the following settings:

- The number of completions in the RX buffer
- The rate at which the Application Layer issues read requests and processes the completion data

Read Request Size

Another factor that affects throughput is the read request size. If a requester requires 4 KB data, the requester can issue four, 1 KB read requests or a single 4 KB read request. The 4 KB request results in higher throughput than the four, 1 KB reads. The read request size is limited by the `Maximum Read Request Size` value in `Device Control` register, bits [14:12].

Outstanding Read Requests

A final factor that can affect the throughput is the number of outstanding read requests. If the requester sends multiple read requests, the number of outstanding read requests is limited by the number of header tags available. The maximum number of header tags is dependent on the **RX Buffer credit allocation - performance for received requests** parameter in the Hard IP for PCI Express IP core **Parameter Editor**.

Document Revision History

Date	Version	Changes
May, 2017	3.0	Made the following changes: • Added Arria 10 reference design. • Updated the "Parameter Settings for Gen3x8 PCI Hard IP variation" section as per each device.
November, 2014	2.0	Made the following changes: • Updated the link to download the reference design and the design software. • Corrected typographical errors in the unit used to express data rate and throughput.
September, 2014	1.0	Initial Release.

Send Feedback