PCI Express DMA Reference Design Using External DDR3 Memory for Stratix V and Arria V GZ Devices

2014.09.19

AN-708 Subscribe Send Feedback

The PCI Express® DMA reference design using external DDR3 memory highlights the performance of the Altera Stratix® V and Arria® V GZ 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 memory controller. It transfers data between an external DDR3 memory and system memory. The reference design includes a Linux-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 PCI Express protocol in using the Avalon-MM 256-bit interface.

Related Information

- **V-Series Avalon-MM DMA Interface for PCIe Solutions User Guide**
- **PCI Express Base Specification Revision 3.0**
- **PCI Express Avalon-MM High Performance DMA with External DDR3 Memory 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 application and driver
- FPGA programming files for the Stratix V GX FPGA Development for Gen3 x8
- Quartus II Archive Files (**. qar**) for the development boards, including SRAM Object File (**. sof**) and SignalTap II Files (**.stp**)

Reference Design Functional Description

The reference design includes the following components:

- An Application Layer which is an Avalon-MM DMA example design generated in Qsys
- The Stratix V Hard IP for PCI Express IP Core
- A Linux software application and driver configured specifically for this reference design

Project Hierarchy

The reference design uses the following directory structures:

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

Stratix V Hard IP for PCI Express Parameter Settings

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	x8
Lane rate	Gen3 (8.0 Gbps)
RX buffer credit allocation – performance for received request	Low
Reference clock frequency	100 MHz
Enable configuration via the PCIe link	Disable
Use ATX PLL	Disable

Table 2: Base Address Register (BAR) Settings

Parameter	Value
BAR0 Type	64-bit prefetchable memory
BAR0 Size	64 KBytes – 16 bits
BAR1 – BAR3	Disable
BAR4	128 MBytes – 27 bits
BAR5	Disable

Table 3: Device Identification Register Settings

Parameter	Value
Vendor ID	0x00001172
Device ID	0x0000E003
Revision ID	0x00000001
Class Code	0x00000000

Parameter	Value
Subsystem Vendor ID	0x00000000
Subsystem Device ID	0x00002861

Table 4: PCI Express/PCI Capabilities

Parameter	Value
Maximum payload size	256 Bytes
Completion timeout range	ABCD
Implement Completion Timeout Disable	Enable

Table 5: Error Reporting Settings

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

Table 6: Link Settings

Parameter	Value
Link port number	1
Slot clock configuration	Enable

Table 7: MSI and MSI-X Settings

Parameter	Value
Number of MSI messages requested	1
Implement MSI-X	Disable
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	32

Quartus II 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

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 memory.

Write Data Mover

The Write Data Mover reads data from the external DDR3 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 embedded into the main DMA to facilitate the customization of descriptor handling. For your application, you can use an external descriptor controller.

Inside the controller, there is a FIFO which is used to store the descriptors which are fetched by the Read Data Mover. The host software programs the internal registers of the Descriptor Controller with the location and size of the descriptor table residing in the PCI Express main memory through the RXM. 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 to transfer the data between the Avalon-MM and PCIe domains one descriptor at a time. It also sends DMA status upstream via the TXS slave port.

TX Slave

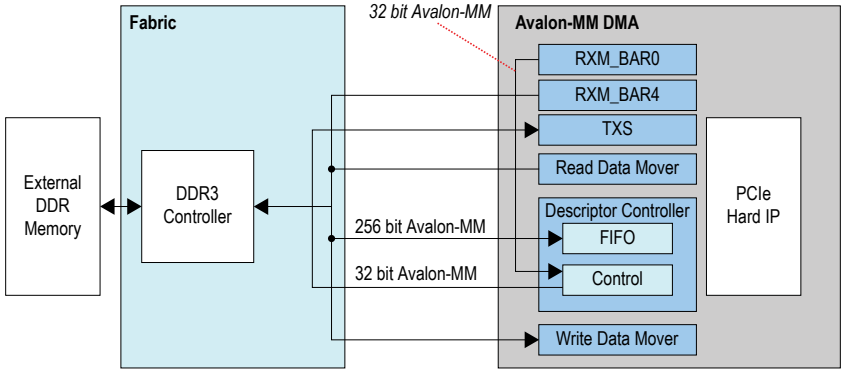
The TX Slave module propagates 32-bit Avalon-MM reads and writes upstream. External Avalon-MM masters, including the DMA control master, can access PCI Express memory space using the TX Slave. The DMA uses this path to update the DMA status upstream, including Message Signaled Interrupt (MSI) status.

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 programs the RX Master to send control, status, and descriptor information to Avalon-MM slave, including the DMA control slave.

Reference Design

Figure 1: DMA Reference Design Block Diagram



The reference design uses an external DDR3 memory with the Altera DDR3 controller that can access up to 128 MB of on-board external DDR3 memory on the application side. The read DMA moves the data from the system memory to the external DDR3 memory. The write DMA moves the data from the external DDR3 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 DDR3 memory is lower compared to the on-chip memory.

Figure 2: DMA Reference Design Qsys Connections

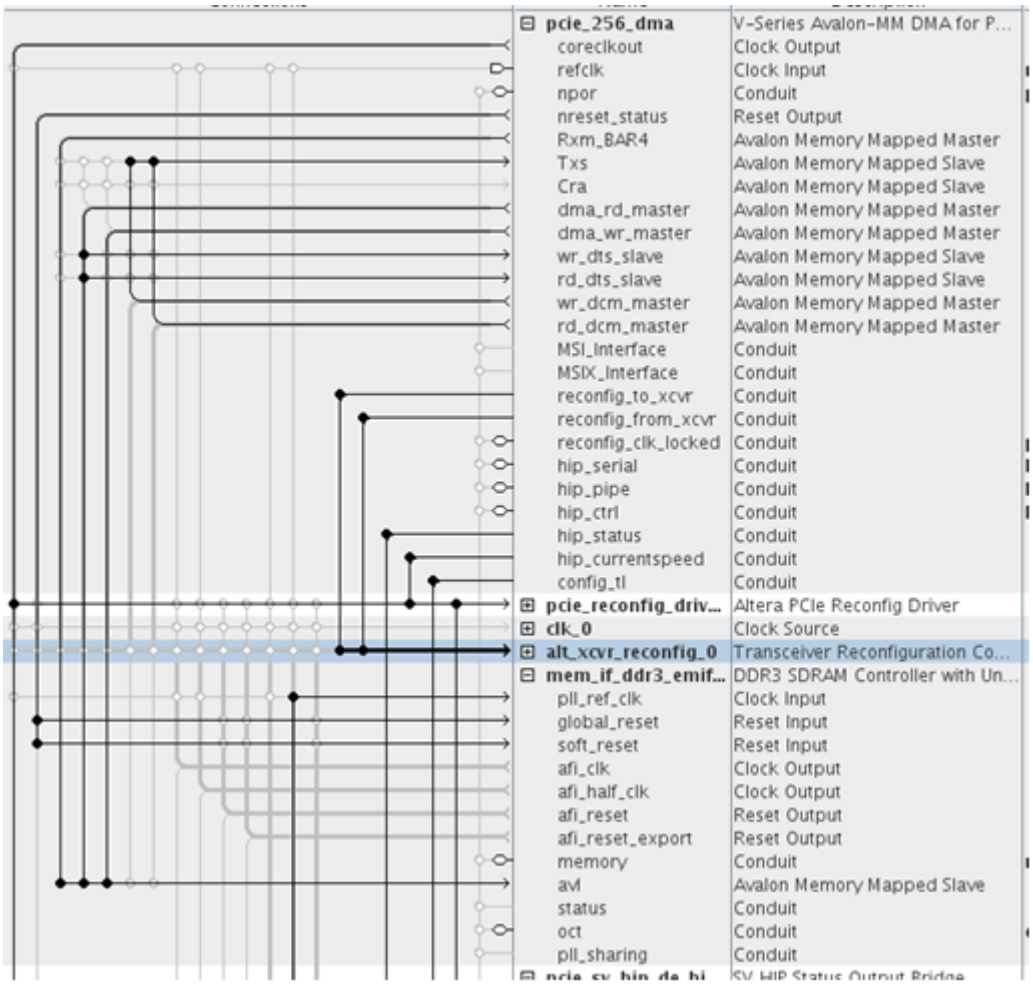


Table 10: AVMM DMA Reference Design Blocks and Port Descriptions

Block	Port	Description
AVMM DMA	`pcie_256_dma`	This is the 256-bit Avalon Memory Mapped module with DMA. It consists of two read and write data movers, a RX master, a TX slave, and an internal descriptor controller. This reference design uses an internal descriptor controller. However, 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.

Block	Port	Description
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 memory. The PCIe host accesses the memory through PCIe BAR4 In a typical application, software controls this port to initialize random data in the external DDR3 memory. Software also reads the data back to verify correct operation.
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 PCIe domain. The descriptor controller uses it to write DMA status back to descriptor data 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 memory through port s2 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 write DMA descriptor FIFO and rd_dts_slave for 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 memory and then writes data to PCIe domain. In this reference design, it uses another port to access the dual port on-chip memory. In this reference design, because the DDR3 controller has a single port, the Write Data Mover uses the same port as the Read Data Mover.

Block	Port	Description
FIFO in Descriptor Controller	wr_dts_slave and rd_dts_slave	This is an Avalon-MM slave port for the FIFO in 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 0x0801_2000—0x0801_3FFF. For the read DMA, the FIFO address is from 0x0801_0000—0x0801_1FFF. When software enumerates the DMA, it generates the descriptor destination addresses.
Control in Descriptor Controller	wr_dcm_master and rd_dcm_master	The control block in Descriptor Controller has two transmit and receive ports. 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 Controller	Altera DDR3 controller	This is a single port DDR3 controller. It can access up to 128 MB external DDR3 memory. In this reference design, the frequency of the DDR3 controller is 500 MHz.

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 space 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 of the data to be moved to by the DMA.
4. For the read DMA operation, the software fills the random data in the system memory space. The Read Data Mover moves this data from the system memory to the external DDR3 memory. For write DMA

operation, the software fills the random data in the external DDR3 memory. The Write Data Mover moves the data from the external DDR3 memory to the system memory space.

5. Software programs the registers in the Descriptor Controller's control logic through PCIe bar0. Programming specifies the base address of the descriptor space which stores the descriptors in the system memory and the base address of the space which is going to store the descriptors in application domain. For this reference design, the base address is 0x0801_1FFF for read DMA and 0x0801_3FFF for write DMA.
6. As the last step to start the DMA, software writes the ID of the last descriptor into the Descriptor Controller's control logic and then triggers the Descriptor Controller to start the DMA. The DMA then starts fetching the descriptors in order from descriptor 0 to the last descriptor.
7. After DMA fetches the last descriptor, the Descriptor Controller writes `1'b1` to the `Done` bit in the last descriptor in the PCIe domain through the `Txs` path.
8. Software polls the `Done` bit in 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 DDR3 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. The operation is complete when both the read and write DMA operation flows finish.

Running the Reference Design

The reference design has the following hardware and software requirements:

Hardware Requirements

- The Stratix V FPGA Development Kit
- A computer running 32 bit or 64 bit Linux with a PCI Express slot. This computer is referred as computer number 1.
- A second computer with the Quartus® II software installed. This computer downloads the FPGA programming file (**.sof**) to the FPGA on the development board. This computer is referred as computer number 2.
- A USB cable or other Altera download cable.

Software Requirements

- The reference design software installed on computer number 1.
- The Quartus II software version 14.0 or later installed on computer number 2.

Installing the Software

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.

Note: 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.
```

You can ignore this message.

Related Information

PCI Express Avalon-MM High Performance DMA with External DDR3 Memory Reference Design
To download the reference design and the design software.

Installing the Hardware

1. Power down computer number 1.
2. Plug the Stratix V FPGA Development Board into a PCI Express slot.
   The development board has an integrated USB-Blaster for FPGA programming.
3. Connect a USB cable from computer number 2 to the Stratix V FPGA Development Board.
4. On computer number 2, bring up the Quartus II programmer and program the FPGA through an Altera USB-Blaster cable.
5. To force the system enumeration to discover the PCI Express IP core, reboot computer number 1.

Running the DMA Software

1. In a terminal window 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 driver, type `./run` .
   The driver prints out the commands available to specify the DMA traffic you want to run. By default, the software enables DMA read, DMA write, 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 DMAs
5	Set the number of dwords per DMA. The legal range is 256-4096 dwords
6	Set the number of descriptors. The legal range is 1-127 descriptors.
7	Set the Root Complex source address offset. This address must be a multiple of 4.
8	Run a loop DMA.

Command	Operation
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.

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

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

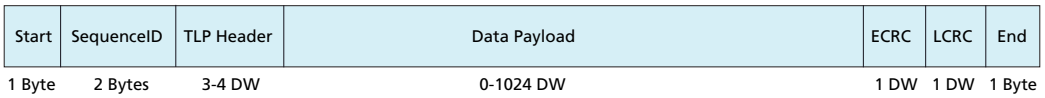
Protocol Overhead

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 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, the ACK/NAK protocol, and flow control DLLPs.
- 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

The figure illustrates the TLP packet format.



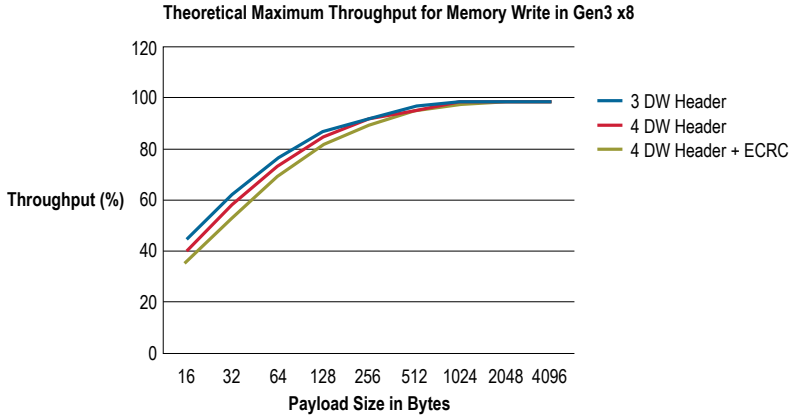
Throughput for Posted Writes

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 Stratix V Hard IP for PCI Express IP Core. You specify this read-only parameter, called **Maximum Payload Size**, in the Stratix V 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, the throughput is limited. 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

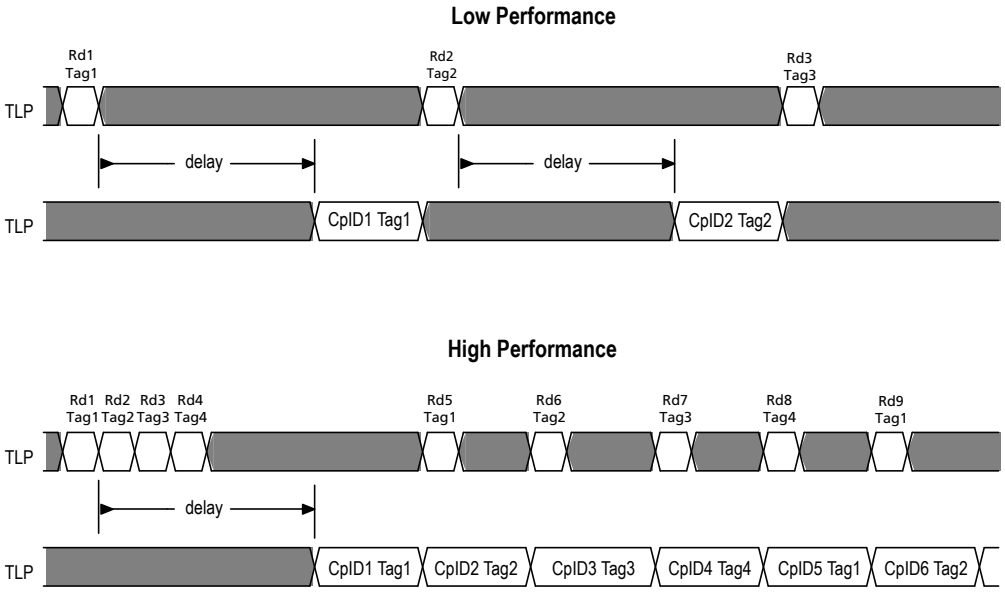
PCI Express uses a split transaction model for reads. The read transaction includes the following steps:

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 taken by the requested data to return. 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 KByte read requests or a single 4 KByte read request. The 4 KByte request results in higher

throughput than the four, 1 KByte 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 specified by 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
September, 2014	1.0	Initial Release.