PCI Express* Avalon®-MM DMA Reference Design

Updated for Intel® Quartus® Prime Design Suite: **Quartus Prime Pro v17.1 Stratix 10 ES Editions**

Online Version
Send Feedback

AN-690

ID: 683824
Version: 2017.05.08

Contents

1. AN 690: PCI Express Avalon-MM DMA Reference Design

The PCI Express® Avalon® Memory-Mapped (Avalon-MM) Direct Memory Access (DMA) Reference Design highlights the performance of the Avalon-MM 256-Bit Hard IP for PCI Express IP Core. The design includes a high-performance DMA with an Avalon-MM interface that connects to the PCI Express Hard IP core. It transfers data between on-chip memory and system memory. The reference design includes both Linux and Windows software drivers that set 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. Measurements show that this design delivers a peak performance of 7.1 giga byte per second (GBps) for reads and writes.

Related Information

- Intel Stratix 10 Avalon-MM Interface for Solutions User Guide PCIe
- Intel Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide
- 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 Reference Design
- PCIe with On-Chip Memory Interface Reference Designs
  To download the reference design, hardware and software packages, follow the steps below:
  1. From the link, expand the **Getting Started - Reference Designs, Application Notes, User Guides & Informational Links** heading.
  2. Browse to the **PCIe with On-Chip Memory Interface Reference Designs** section.
  3. Click on the **Altera Wiki** link for your development kit.
  4. On the Altera Wiki page, browse to the **Download** section to download the appropriate hardware and software packages.

1.1. Deliverables Included with the Reference Design

The reference design includes the following components:

- Linux and Windows applications and drivers
- FPGA programming files for the Arria® V , Arria 10, Cyclone® V, Stratix® V, or Stratix 10 FPGA Development Boards.
- Quartus® Prime Archive Files (`.qar`) for the development boards, including an SRAM Object File (`.sof`) and Signal Tap files (`.stp`).

1.2. 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
- An Arria V, Arria 10, Cyclone V, Stratix V, or Stratix 10 Hard IP for PCI Express IP Core
- A Linux or Windows 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.

1.2.1. 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	Arria 10, Stratix V, and Stratix 10: x8 Arria V and Cyclone V®: x4
Lane rate	Arria 10 , Stratix V, and Stratix 10: Gen3 Arria V and Cyclone V®: Gen2
RX buffer credit allocation – performance for received request	Arria V, Arria 10, Cyclone V, and Stratix V: Low Stratix 10: Not available
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 KB – 16 bits
BAR4 Type	64-bit prefetchable
BAR4 size	64 KB – 16 bits
BAR1-3, BAR5	Disable

Table 3. Device Identification Register Settings

Parameter	Value
Vendor ID	0x00001172
Device ID	0x0000E003
Revision ID	0x00000001
Class Code	0x00000000
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	4
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 Prime Settings

The Quartus Prime Archive File (`*.qar`) file in the reference design package has the recommended synthesis, fitter, and timing analysis settings for the parameters specified in this reference design.

1.3. Avalon-MM DMA

The Avalon-MM interface with DMA includes the following modules:

Read Data Mover

The Read Data Mover sends memory read TLPs upstream. After the Completion is received, it writes the received data to the on-chip memory.

Write Data Mover

The Write Data Mover reads data from on-chip 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, a FIFO stores the descriptors that the Read Data Mover fetches. 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 RX Avalon-MM 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 to transfer the data between the Avalon-MM and PCIe domains one descriptor at a time. It also sends DMA status upstream via the TX Avalon-MM slave port.

TX Slave

The TX Avalon-MM 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.

1.4. Reference Design

The reference design uses a 64-KB dual port memory in the FPGA in the FPGA fabric. The read DMA moves the data from the system memory to the 64-KB dual port memory. The write DMA moves the data from the 64-KB dual port memory to the system memory. Because this is a dual port memory, the read and the write DMA can access memory simultaneously.

Figure 1. DMA Reference Design Block Diagram

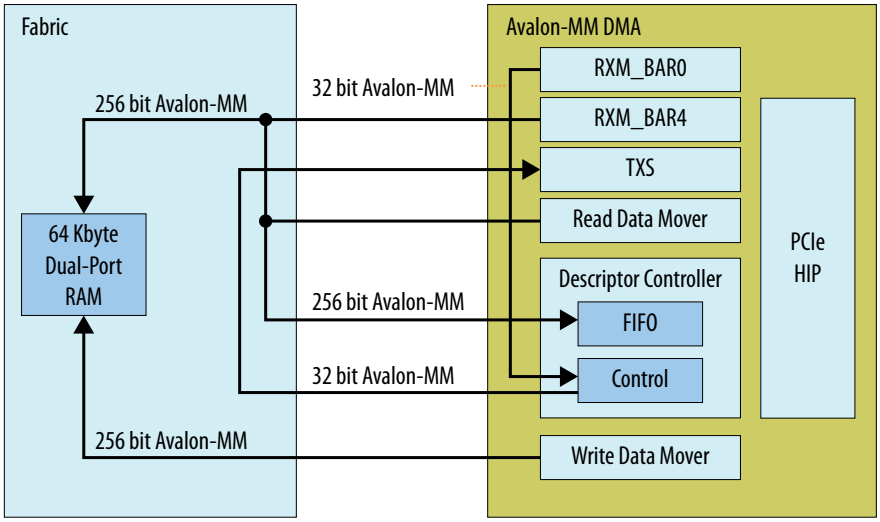


Figure 2. DMA Reference Design Qsys® Connections

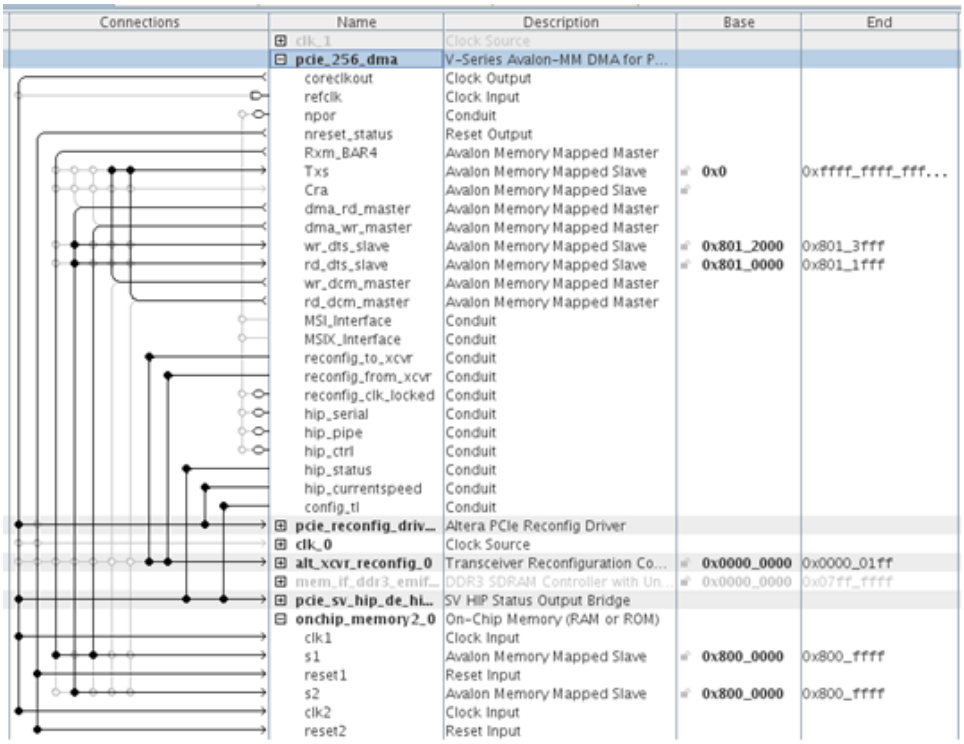


Table 10. AVMM DMA Reference Design Blocks and Port Descriptions

Block	Port	Description
Avalon-MM DMA	`pcie_256_dma`	This is the 256-bit Avalon-MM with DMA module. It consists of two read and write data movers, a RX Avalon-MM master , a TX Avalon-MM slave, and an internal descriptor controller. This reference design uses an internal descriptor controller. You can also specify an external descriptor controller using the parameter editor.
RXM_BAR0	N/A	This is an Avalon-MM master port. It provides memory access to the PCIe host. The host programs the descriptor controller through this port Because this reference design uses an internal descriptor controller, the port connection is not shown in Qsys. The connection is made inside the Avalon-MM DMA module.
RXM_BAR4	`Rxm_BAR4`	This is an Avalon-MM master port. It provides memory access to the PCIe host. In the reference design, connects to one port of the 64-KB on-chip memory. In a typical application, software controls this port to initialize the on-chip memory and read back the data to verify results.
TX Avalon-MM Slave	`Txs`	This is an Avalon-MM slave port. In a typical application, an Avalon-MM master drives this por to send memory reads or writes to the PCIe domain. The descriptor controller uses it to write DMA status back to descriptor data in the PCIe domain when the DMA completes. 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.

continued...

Block	Port	Description
		The Read Data Mover moves data from the PCIe domain to the on-chip memory through port s2 during normal read DMA operations. The Read Data Mover also fetches the descriptors from the PCIe domain and writes them to the FIFO in the Descriptor Controller. The `dma_rd_master` port connects to the write DMA FIFO, `wr_dts_slave`, and read DMA FIFO, `rd_dts_slave`, ports.
Write Data Mover	`dma_wr_master`	This is an Avalon-MM master port. The Write Data Mover generates reads to read data from on-chip memory and then writes the data to a destination in PCIe domain. In this reference design, it uses another port to access the dual port on-chip memory.
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 through this port. Because there are two separate groups of descriptors for read and write, there are two ports available. For the write DMA, the FIFO address range is `0x0801_2000 -0x0801_3FFF`. For the read DMA, the FIFO address range is `0x0801_0000 -0x0801_1FFF`. Software enumeration the DMA establishes these address.
Control in Descriptor Controller	`wr_dcm_master` and `rd_dcm_master`	The Descriptor Controller's control block has two transmit and receive ports. One for read DMA and another one for write DMA. The receive port connects to `RXM_BAR0`. The transmit port connects to `Txs`. The receive path from `RXM_BAR0` is connected internally, so 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 appear in the *DMA Reference Design Qsys Connections* figure.
64k-byte Dual Port RAM	`Onchip_memory2_0`	This is a 64-KB dual port on-chip memory. The address range is `0x0800_0000`- `0x0800_FFFF` on the Avalon-MM bus. This address is the source address for write DMAs or destination address for read DMAs. To prevent data corruption, software divides the memory into separate regions for reads and writes. The regions do not overlap.

1.5. DMA Operation Flow

Software running on the host completes the following steps to initiate the DMA and verify the results.

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 of the data to be moved to by the DMA.

4. For the read DMA operation, the software initializes the random data in the system memory space. The Read Data Mover moves this data from the system memory to the on-chip memory. For write DMA operation, the software initializes the random data in the on-chip memory. The Write Data Mover moves the data from the on-chip memory to the system memory space.
5. As the last step to start the DMA, software writes the ID of the last descriptor into the Descriptor Controller's control logic. This write triggers the Descriptor Controller to start the DMA. The DMA then starts fetching the descriptors in order from descriptor 0 to the last descriptor.
6. After the last descriptor completes, the Descriptor Controller writes a "1" to the `Done` bit in the last descriptor in the PCIe domain through the `Txs` port.
7. Software continues polling the `Done` bit in the last descriptor. The DMA operation is said to be completed when the `Done` bit is set and the performance number is calculated. Once the DMA completes, Software compares the data in system memory and the on-chip memory. The test passes if there is no data mismatch.
8. For simultaneous operation, the software starts the read DMA operation followed by the write DMA. The operation is considered done after both the read and write DMA operation flows complete.

1.6. Running the Reference Design

The reference design has the following hardware and software requirements:

Hardware Requirements

- The 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 or a computer running 64-bit Windows. This computer downloads the SRAM Object File (`*.sof`) to the FPGA on the development board. This computer is referred as computer number 2.
- A USB cable or other download cable.

Software Requirements

- The reference design software installed on computer number 1.
- The Quartus Prime software installed on computer number 2.
  - Arria V, Cyclone V®, and Stratix V: require Quartus Prime Standard Edition, version 15.1 or later
  - Arria 10: requires Quartus Prime Standard or Pro Edition, version 16.0 or later
  - Startix 10: requires Quartus Prime Pro Edition, version 17.1IR1 or later

Related Information

- PCI Express Avalon-MM High-Performance DMA Reference Design for Arria V
  To download the reference design and for more details.
- PCI Express Avalon-MM High-Performance DMA Reference Design for Arria 10
  To download the reference design and for more details.

- PCI Express Avalon-MM High-Performance DMA Reference Design for Cyclone V
  To download the reference design and for more details.
- PCI Express Avalon-MM High-Performance DMA Reference Design for Stratix V
  To download the reference design and for more details.
- PCI Express Avalon-MM High-Performance DMA Reference Design for Stratix 10
  To download the reference design and for more details.
- PCI Express Protocol

1.6.1. 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.
```

1.6.2. 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.

1.6.3. Installing the Hardware

1. Power down computer number 1.
2. Plug the FPGA Development kit into a PCI Express slot.

   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.

1.6.4. Running the Software

Complete the following steps to run 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 allowed range is 256-4096 dwords
6	Set the number of descriptors. The allowed 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.
10	Exit.

7. To start the DMA, type 1. This 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 1.1                                 **
** 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) set rc source addr offset (mult of 4)    **
** 8) loop dma                                 **
** 10) exit                                    **
*************************************************
Run Read                 ? 1
Run Write                ? 1
Run Simultaneous         ? 1
Read Passed              ? 1
Write Passed             ? 1
Simultaneous Passed      ? 1
Number of Dwords/Desc    : 4096
Number of Descriptors    : 127
Length of transfer       : 2032 KB
Rootport address offset  : 4
Read Time                : 0 s and 299 us
Read Throughput          : 6.483371 GB/S
Write Time               : 0 s and 304 us
Write Throughput         : 6.376737 GB/S
Simultaneous Time        : 0 s and 337 us
Simultaneous Throughput  : 5.752309 GB/S
#
```

Understanding Throughput Measurement

To measure throughput, the software driver takes two timestamps. Software takes the first timestamp shortly after the you type the `Start DMA` command. Software takes the second timestamp after the DMA completes and returns the required completion status, EPLAST. If read DMA, write DMA and simultaneous read and write DMAs are all enabled, the driver takes six timestamps to make the three time measurements.

This DMA performance testing used 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

1.6.5. Peak Throughput

Signal Tap calculates the following throughputs for the Gen3 x8, 256-bit DMA running at 250 MHz for 142 cycles:

- **7.1 GBps**-for TX Memory Writes only, 256-byte payload, with the cycles required for transmitting read requests ignored in calculation
- **6.8 GBps**- for TX Memory Writes, 256-byte payload, with the cycles required for transmitting read requests included in calculation
- **7.0 GBps**-for RX Read Completions

In the Signal Tap run, the `TxStValid` signal never deasserts. The Avalon-MM application drives TLPs at the maximum rate with no idle cycles. In addition, the Hard IP for PCI Express continuously accepts the TX data. The Hard IP for PCI Express IP Core rarely deasserts the `RxStValid` signal. The de-assertions of the `RxStValid` signal occur because the host system does not return completion data fast enough. However, the Hard IP for PCI Express IP Core and the Avalon-MM application logic are able to handle the completions continuously as evidenced by long the stretches of back-to-back TLPs.

The average throughput for 2 MB transfers including the descriptor overhead is 6.4 GB/s in each direction, for a total bandwidth of 12.8 giga byte per second (GBps). The reported DMA throughput reflects typical applications. It includes the following overhead:

- The time required to fetch DMA Descriptors at the beginning of the operation
- Periodic sub-optimal TLP address boundaries allocation by the host PC
- Storing DMA status back to host memory
- Credit control updates from the host
- Infrequent Avalon-MM core fabric stalls when the `waitrequest` signal is asserted

TX Theoretical Bandwidth Calculation

For TX interface, the write header is 15 bytes. The read header is 7 bytes. Because the TX interface is 256 bits, transferring either RX read and write headers requires 1 cycle. Transferring 256 bytes of payload data requires 8 cycles (256/32), for a total of 9 cycles per TLP. The following equation calculates the theoretical maximum TX bandwidth:

```
8 data cycles/9 cycles x 8 GBps = 7.111 GBps
```

Consequently, the header cycle reduces the maximum theoretical bandwidth by approximately 12%.

Signal Tap Observation

Signal Tap simulations show a peak throughput of 142 cycles at 250 MHz to transfer 2 MB of data. Disregarding the time required for Read Requests in the overhead calculation, results in the following equation:

```
(142 - 7 - 15)/(142 - 7) x 8 GBps) = 7.111 GBps
```

Including the time required for Read Requests in the overhead calculation, results in the following equation:

```
(142 - 7 - 15)/(142 x 8 GBps) = 6.76 GBps
```

RX Theoretical Bandwidth Calculation

For the RX interface, for a 2 MB transfer, the `RxStValid` signal is not asserted for 3 cycles. The transfer requires 15 Read Completion headers for 256-byte completions. For each Read Completion, the maximum RX bandwidth is represented by the following equation:

```
8 data cycles/9 cycles * 8GBps = 7.1111 GBps
```

Signal Tap Observation

Including the number of Completion headers and cycles when the `RxStValid` signal is not asserted as overhead, the actual observed bandwidth is represented by the following equation:

```
(142-15-3)/142 * 8GBps = 6.986 Gbps
```

1.7. Understanding PCI Express Throughput

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

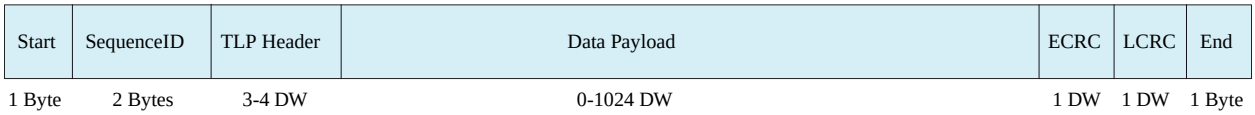
- Protocol overhead
- Payload size
- Completion latency
- Flow control update latency
- Devices forming the link

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



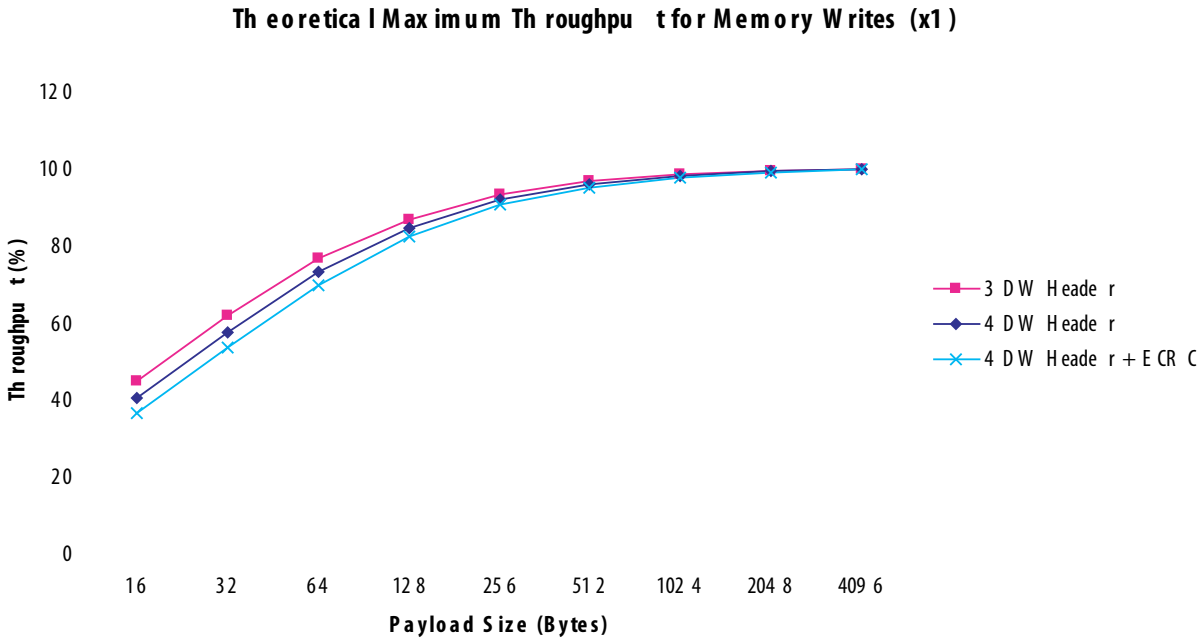
1.7.1. Throughput for Posted Writes

The theoretical maximum throughput is calculated using the following formula:

```
Throughput  = payload size / (payload size + overhead) * link data rate
```

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. For a 256-byte maximum payload size and a 3-dword header the overhead is five dwords. Because the interface is 256 bits, the 5-dword header requires a single bus cycle. The 256-byte payload requires 8 bus cycles.



The following equation shows maximum theoretical throughput:

```
Maximum throughput = 8 cycles/9 cycles  = 88.88% * 8 GBps = 7.2 GBps
```

1.7.1.1. 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. You specify this read-only parameter, called **Maximum Payload Size**, in the Avalon-MM 256-Bit Hard IP for PCI Express GUI. 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 by the rate that 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 two connected devices.

1.7.2. Throughput for Reads

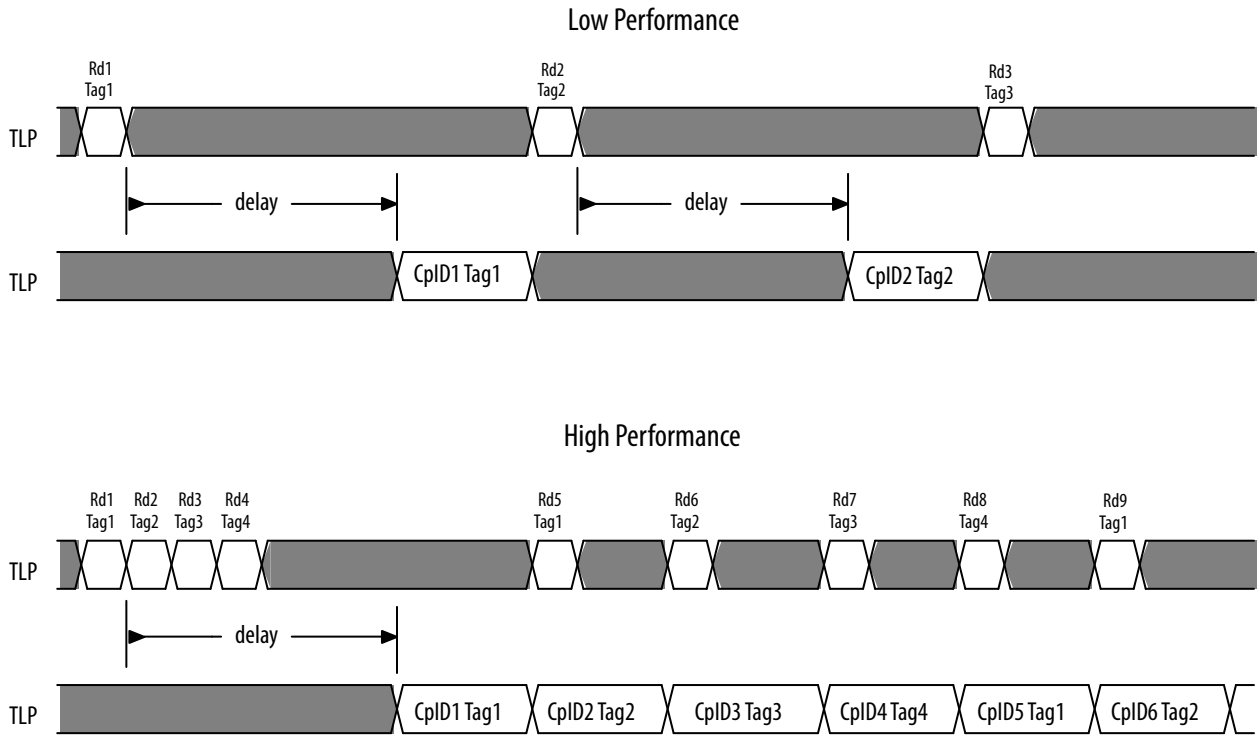
PCI Express uses a split transaction model for reads. The read transaction includes the following steps:

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 also depends on the round trip delay between the time when the Application Layer issues a Memory Read Request and the time when the requested data returns. 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 to improve throughput, the number of outstanding read requests is limited by the number of header tags available. To achieve high performance, the read DMA can use up to 16 header tags.

1.8. Document Revision History for AN-690

Date	Version	Changes
May 2017	Quartus Prime Pro v17.1 Stratix 10 ES Editions	Made the following changes: • Added support for Stratix 10 devices. • Fixed minor errors and typos.
May 2016	6.0	Made the following changes: • Added support for Linux.
February 2016	5.0	Made the following changes: • Added support for Windows. • Added support for Arria V , Arria 10, and Cyclone V devices.
Novermber, 2014	4.0	Corrected typographical errors in the unit used to express data rate and throughput.
September, 2014	3.0	Updated the link in the introduction section to point to the latest version of the *V-Series Avalon-MM DMA Interface for PCIe Solutions User Guide*.
July, 2014	2.0	Made the following changes: • Updated the *Avalon-MM DMA* section to indicate that descriptor controller is embedded in the main DMA. Updated the descriptions of Read Data Mover, Write Data Mover, and Descriptor Controller. • Updated *DMA Operation Flow* section. • Removed *DMA Descriptor Registers* and *DMA Control and Status Registers* sections. • Added a new section called *Reference Design* with block diagram, block, and port descriptions. • Updated the steps in *Software Installation* and *Running the Software* sections.
August, 2013	1.0	Initial Release.