Introduction

PCI Express is a high-performance, general purpose I/O interconnect defined for a wide variety of computing and communication platforms. PCI Express uses a serial point-to-point packetized interface while maintaining software compatibility with the older PCI protocol. PCI Express provides higher performance and reduces overall system cost compared to the multi-drop parallel PCI and PCI-X buses.

Altera’s PCI Express Compiler generates customized high performance PCI Express MegaCore® functions that you can use to design PCI Express endpoints. The PCI Express MegaCore functions are compliant with PCI Express Base Specification Revision 1.1 and PCI Express Base Specification Revision 1.0a.

The PCI Express high-performance reference design uses the Altera® PCI Express MegaCore and includes a high-performance chaining direct memory access (DMA) that transfers data between the FPGA internal memory and the system memory. The reference design includes a Windows XP-based software application that sets up the DMA transfers. The software application also measures and displays the performance achieved for the transfers. The chaining DMA used in this reference design is the chaining DMA example generated by the PCI Express Compiler. This reference design highlights the performance of the Altera PCI Express MegaCore and enables you to evaluate the PCI Express MegaCore.

This application note contains the following information:

- “Understanding Throughput in PCI Express” on page 2
- “Deliverables Included with the Reference Design” on page 6
- “Reference Design Functional Description” on page 6
- “Design Walkthrough” on page 11
- “Performance Benchmarking Results” on page 21
Understanding Throughput in PCI Express

The throughput in a PCI Express system depends on several factors, including protocol overhead, payload size, completion latency, and flow control update latency. The throughput also depends on the characteristics of both devices in the link. This section discusses the various factors that you need to consider when analyzing throughput. This example assumes an x1 link operating at 2.5 Gbps.

Protocol Overhead

PCI Express uses 8b/10b encoding, in which every byte of data is converted into a 10-bit data code, resulting in a 25% overhead. The effective data rate is therefore reduced to 2 Gbps or 250 MBps per lane.

An active link is also used to transmit Data Link Layer Packets (DLLPs) and Physical Layer Packets (PLPs). The PLPs are four bytes or one double word (DW) in size and consist of SKP ordered sets. The DLLPs are two DWs in size and consist of the ACK/NAK and flow control DLLPs. The ACKs and flow control update DLLPs are transmitted in the opposite direction from the Transaction Layer Packet (TLP). In cases where the link is being driven with high bandwidth traffic in both directions, the DLLP activity can be significant and on the order of one DLLP for every TLP. The DLLPs and PLPs reduce the effective bandwidth available for TLPs. The TLP format is shown in Figure 1.

Figure 1. TLP Format

<table>
<thead>
<tr>
<th>Start</th>
<th>Sequence ID</th>
<th>Header</th>
<th>Data Payload</th>
<th>ECRC</th>
<th>LCRC</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>2 Bytes</td>
<td>3-4 DW</td>
<td>0-1024 DW</td>
<td>1 DW</td>
<td>1 DW</td>
<td>1 Byte</td>
</tr>
</tbody>
</table>

The overhead associated with a single TLP varies between five and seven DWs. The overhead includes the Start and End framing symbols, the Sequence ID, a TLP header that is three or four DW long, an optional ECRC, and the LCRC. The TLP header size depends on the TLP type and can change from one TLP to another. The rest of the TLP contains 0–1024 DW of data payload.

Throughput for Posted Writes

The theoretical maximum throughput is calculated using the following formula:

Throughput % = payload size / (payload size + overhead)
Figure 2 shows the maximum throughput possible with different TLP header sizes and ignores any DLLPs and PLPs. For a 256-byte maximum payload size and a three DW TLP header (or five DW overhead), the maximum possible throughput is \((256/(256+20))\), or 92%.

**Figure 2. Maximum Throughput for Memory Writes**

The maximum TLP payload size is controlled by the Device Control Register (bits 7:5) in the PCI Express Configuration space. The MegaCore parameter **maximum payload size supported** sets the read-only value of the **maximum payload size supported** field of the Device Capabilities Register (bits 2:0) and optimizes the MegaCore function for this payload size. The MegaCore function can be configured for a maximum payload size, which can then be reduced by the system based on the maximum payload size supported by the system. This MegaCore parameter affects the core utilization, so the parameter must be set such that it is not greater than the maximum payload size supported by the system in which the core is used.

PCI Express uses flow control, in which a TLP is not transmitted unless the receiver has enough free buffer space to accept that TLP. A device needs sufficient header and data credits before sending a TLP. When the application logic 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 (FC Update DLLP) that returns the credits consumed by the originating TLP. When the device uses up all of its initial credits,
the link bandwidth is limited by how fast it receives credit updates. Flow control updates depend on the maximum payload size and the latencies in the transmitting and receiving devices.

For more information about the flow control update loop, refer to the *PCI Express Compiler User Guide*. This guide provides detailed information about the flow control update loop and the associated latencies.

**Throughput for Reads**

PCI Express uses a split-transaction for reads. A requester first sends a memory read request. The completer then sends an ACK DLLP to acknowledge the memory read request. It subsequently returns a completion data that can be split into multiple completion packets. Read throughput is significantly lower than write throughput because the completions can be split into multiple packets, as illustrated by the following example. Assuming a read request for 512 bytes and a completion packet size of 64 bytes, the maximum possible throughput can be calculated as follows:

\[
\text{Number of completion packets} = \frac{512}{64} = 8
\]

\[
\text{Overhead for a 3 DW TLP Header with no ECRC} = 8 \times 20 = 160 \text{ bytes}
\]

\[
\text{Maximum Throughput} = \frac{512}{512 + 160} = 76\%.
\]

These calculations do not take into account any DLLPs and PLPs. The Read Completion Boundary (RCB) parameter specified by the PCI Express specification determines the naturally aligned address boundaries on which a Read Request may be serviced with multiple completions. For a Root Complex, the RCB is either 64 bytes or 128 bytes. For all other PCI Express devices, the RCB is 128 bytes.

A non-aligned read request may experience a further throughput reduction.

Read throughput depends on the round-trip delay between when the application logic issues a read request and when all of the completion data has been returned. The throughput depends on the application being able to issue enough read requests and the completer processing (or at least offering enough non-posted header credits) to cover this delay.

Figure 3 shows the time line for memory read requests (MRd) and completions (CplD). The time line in the top part of the figure shows the requester waiting for a completion before making a subsequent read
request, resulting in a lower throughput. The time line in the bottom part shows the requester making enough memory read requests to cover the delay for the completions, thus maintaining a higher throughput.

In addition, the requester must maintain maximum throughput for the completion data packets. This can be done by selecting appropriate settings for completions in the Rx buffer and by managing the rate at which the application logic issues read requests and processes the completion data.

For more recommendations for the Rx buffer settings, refer to the PCI Express Compiler User Guide.

A final constraint on the throughput is the amount of read request data that can be outstanding at one time. This amount is limited by the number of header tags and the maximum read request size that can be issued. The maximum read request size is controlled by the Device Control Register.
(bits 14:12) in the PCI Express Configuration space. The header tag is a number that is given to a non-posted request by the Application layer to distinguish completions for that request from other requests. The maximum number of tags that the application will use needs to be set in the IP Core configuration, because the core tracks read requests and completions for error checking. Specifying a large number of tags increases the resources used by the core and so can decrease the core performance ($f_{\text{MAX}}$). On a Windows system, eight tags are usually enough to ensure continuous read completion with no gap for a 4-Kbytes read request.

### Deliverables

**Included with the Reference Design**

The reference design includes the following components:

- Software application and windows drivers (32-bit, Windows XP)
- FPGA programming files for the Stratix® II GX PCIe development kit for x1, x4, and x8 operation
- Quartus® II Archives (QAR) for the development board and configurations, including SOF, POF, and SignalTap® II files

### Reference Design

**Functional Description**

The reference design consists of the following components:

- An application layer logic that consists of the chaining DMA example generated by the MegaCore function
- The MegaCore function variation
- Software application and Windows XP drivers

These components are shown in Figure 4.

#### Figure 4. Reference Design Components

The chaining DMA example consists of two DMA modules in the application logic and an internal memory (referred to as the Endpoint Memory). The design can support simultaneous DMA read and DMA
write transactions. The DMA Write module implements write operations in which the data is transferred from the endpoint memory to the Root Complex (system memory) across the PCI Express link. The DMA Read module implements read operations in which data is transferred from the Root Complex (system memory) across the PCI Express link to the endpoint memory.

The reference design itself is completely contained within the FPGA and relies on no other hardware interface except the PCI Express link. A chaining DMA provides higher performance compared to a simple DMA when transferring a large amount of non-contiguous memory between the system memory and endpoint memory. In a simple DMA, the software application needs to program the DMA registers for every descriptor. The chaining DMA uses descriptor tables for each transfer. These descriptor tables contain the following information:

- Transfer length
- Source and destination addresses for the transfer
- Control information that sets the handshaking behavior between the software application and the DMA module

Each descriptor consists of four double-words. The descriptors are staggered in a contiguous memory page.

Based on the attributes set in the GUI, the software application creates the necessary descriptor tables in the system memory. The software application also creates a descriptor header table that includes such information as the total number of descriptors and the address of the first descriptor table. At the beginning of the transfer, the software application programs the DMA registers with the descriptor header table. The DMA module continuously collects these descriptor tables for each DMA read and DMA write and performs the transfer for each descriptor.

The DMA module also includes a performance counter. The counter starts when the software writes a descriptor header table to the DMA registers and continues counting until the last data has been transferred by the DMA module. After the transfer is complete, the software application uses the counter value to compute the throughput for the transfer and reports it. The counter value includes latency for the initial descriptor read, so the throughput reported by the software application is less than the actual amount.

For more information about the chaining DMA example architecture and programming instructions, refer to the *PCI Express Compiler User Guide*. 
Core Settings

The cores used in the reference design support a maximum payload size of 256 bytes. The desired performance for received completions and requests is set to High, as shown in Figure 7 on page 10.

Figures 5 to 7 show the core parameters used in the reference design, as set in the MegaWizard® Plug-In Manager. The figures show the core settings for an x8 operation. The settings for x1 and x4 operations are identical except that the number of lanes is set to 1 or 4.

For more information about the core parameters, refer to the PCI Express Compiler User Guide.

Figure 5. MegaWizard Plug-In Manager – System Settings
Figure 6. MegaWizard Plug-In Manager – Capabilities
Figure 7. MegaWizard Plug-In Manager – Buffer Setup

Quartus II Settings

The Quartus II Archives (QAR) included in the reference design package has the recommended synthesis, Fitter, and timing analysis settings for the parameters chosen in the variation used in this reference design.
This section describes how to install the reference design and provides instructions for running the software application. The following information is included:

- Hardware requirements
- Software requirements
- Software installation
- Hardware installation
- Running the software application

### Hardware Requirements

The reference design requires the following hardware:

- Stratix II GX PCI Express development board
- Computer running 32-bit Windows XP with an x8/x4/x1 PCI Express slot for the Stratix II GX PCI Express development board. The software application and hardware are installed on this computer (referred to as computer #1 in this document).
- Computer with the Quartus II software for downloading FPGA programming files to the Stratix II GX PCI Express development board (referred to as computer #2 in this document).
- Altera USB-Blaster™ cable or other Altera download cable
- PCI Express x8-to-x4 lane converter for x4 operations or PCI Express x8-to-x1 lane converter for x1 operations. Lane converters are extender cards that have higher channel connectors and lower channel PCB gold fingers.

PCI Express allows “Up-plugging,” in which a card with fewer lanes can be plugged into larger connectors that can accommodate more lanes. A card with more lanes cannot be plugged into a slot with fewer lanes (“Down-plugging”). This is prevented by the connector keying. Therefore, if you want to plug the Stratix II GX card into an x4/x1 slot, you need to use an x4/x1 lane converter.

### Software Requirements

To run the reference design application requires installation of the following software:

- The Quartus II software version 7.1 running on computer #2.
Software Installation

You need Administrator privileges to install the software application.

The software application only runs on 32-bit Windows XP and uses the WinDriver version 8.11 from Jungo.

Perform the following steps to install the software application and windows drivers:

1. Download the PCI Express High Performance reference design package to computer #2, and extract the compressed files. Figure 8 shows the directory structure.

   ![Figure 8. Directory Structure](image)

2. Without the card plugged in, copy the Software Application directory to computer #1.

3. In the JungoDrivers directory, double-click on install.bat to install the Windows XP driver for this application.

4. Run altpcie_demo.exe from the Software Application directory to run the software application.

Hardware Installation

Perform the following steps to install the hardware:

1. Power down computer #1 and plug the Stratix II GX card into the PCI Express slot. For an x1/x4 operation, use a PCI Express lane converter.

2. Start computer #1.

3. Program the FPGA with the reference design. For best results use a POF (Programming Object File).
Programming the development board is done through the Quartus II software on computer #2, and requires an Altera USB-Blaster cable (or other download cable) connection between computer #2 and the development board on computer #1. The following procedures describe programming the FPGA with the POF or SOF (Software Object File).

- “Connecting the USB-Blaster Cable” on page 13
- “Programming with the POF” on page 13
- “Programming with the SOF” on page 16

**Connecting the USB-Blaster Cable**

To connect the USB-Blaster cable, perform the following steps:

1. Connect the 10-pin female plug of the USB-Blaster cable to the JTAG header (J5) on the Stratix II GX EP2SGX90 PCI Express development board. Connect the marker line on the cable to pin 1 of the header (J5). Pin 1 of J5 is numbered on the board.

2. Connect the other end of the cable to the USB port on the computer running the Quartus II software, computer #2.

**Programming with the POF**

Perform the following steps to program the FPGA with the POF:

1. Start the Quartus II programmer on computer #2.

2. In the Quartus II programmer, click **Auto Detect** to list the devices attached to the JTAG chain on the Stratix II GX board. The result of the operation is shown in **Figure 9**.
Depending on the programming status of the Flash on your board, you might see different results.

3. If the MAX II device has a flash device attached to it, go to step 5. Otherwise, select the MAX II device. Right-click and click **Attach Flash Device**.

4. In the **Attach Flash Device** dialog box, select a CFI_512MB device. Click **OK**.

5. Select the MAX II device (EPM570). Right-click and click **Change File**. In the **Change File** dialog box, select the **pfl2.pof** programming file from the **sof** directory.

6. Click **Open**. The dialog box closes.
7. Select the flash device, right-click and click **Change File**. In the **Change File** dialog box, select the appropriate POF (x1, x4, or x8) from the sof directory.

8. Click **Open**. The dialog box closes.

9. Turn on the Program/Configure box for the MAX II device and the flash device. Click **Start**. The MAX II device and the flash device are programmed with the reference design.

   If a message appears that states “Device 1 silicon ID is not ready” (Figure 10), press the **Config_Done** button (S1) on the development board to start the programming. This issue occurs with boards that have already had their flash programmed.

![Figure 10. Device Not Ready Message](image)

10. After the programming is complete, power down computer #1 and restart it.

11. The operating system detects a new hardware device and displays the **Found New Hardware Wizard**. In the wizard, select **Install the software automatically (Recommended)**. Click **Next**.

12. Click **Finish** to close the wizard.
**Programming with the SOF**

To simplify the use of this software application and avoid the multi-stage boot sequence, use the POF. However, if you want to program the board with the SOF provided, perform the FPGA programming when the Windows Boot sequence is in one of the following safe states:

- When you get an error with a message similar to “Alert! Error initializing PCI Express slot 1”
- During the boot sequence, press the F8 key to start the Windows Boot manager.

Perform the following steps to program the FPGA with the SOF:

1. Start the Quartus II programmer on computer #2.

2. In the Quartus II programmer, click **Auto Detect** to list the devices attached to the JTAG chain on the Stratix II GX board. The result of the operation is shown in Figure 9 on page 14.

3. Select the Stratix II GX device (EP2SGX90). Right-click and click **Change File**. In the **Change File** dialog box, select the appropriate SOF (x1, x4, or x8) from the **sof** directory.

4. Click **Open**. The dialog box closes.

5. Turn on the **Program/Configure** box for the Stratix II GX device.

6. Click **Start**. The Stratix II GX device is programmed with the reference design.

7. Do a soft reboot (Ctl-Alt-Delete, not a power down) on computer #1.

8. The operating system detects a new hardware device and displays the **Found New Hardware Wizard**. In the wizard, select **Install the software automatically (Recommended)**. Click **Next**.

9. Click **Finish** to close the wizard.

**Running the Software Application**

Perform the following steps to run the software application:

1. Double-click on the application **altpcie_demo.exe** in the **Software Application** directory to run it.
2. The application reports the board type, the number of active lanes, the maximum read request size, and the maximum payload size, as shown in Figure 11.

Figure 11. Initial Settings

![Initial Settings](image)

The software GUI has the following control fields:

- **Transfer length**—Specifies the transfer length in bytes
- **Sequence**—Controls the sequence for data transfer
- **Number of iterations**—Controls the number of iterations for the data transfer
- **Board**—Specifies the development board for the software application
- **Continuous loop**—When this option is turned on, the application performs the transfer continuously

3. Set the **Transfer length** to 100,000 bytes and the **Sequence** to **Write only**, as shown in Figure 12. Click **Run**.
The software programs the DMA registers in the FPGA to transfer data from the FPGA to the system memory in chunks of 100,000 bytes. The performance bars report the peak, average, and last throughput. The average throughput is computed across all the iterations.

4. Set the **Transfer length** to 100,000 bytes and the **Sequence** to **Read only**, as shown in Figure 13. Click **Run**.
The software programs the DMA registers to transfer data from the system memory to the FPGA in chunks of 100,000 bytes. The performance bars report the peak, average, and last throughput. The average throughput is computed across all the iterations.

5. Set the **Transfer length** to 100,000 bytes and the **Sequence** to **Read and Write**, as shown in Figure 14. Click **Run**.
In this sequence the software application programs the DMA registers to perform simultaneous read and write transfers. The performance bars show the write throughput and the text shows the average read and write throughput across all the iterations.

**SignalTap II Files**

The reference design package also includes SignalTap II (STP) files that show the operation of the chaining DMA application logic. The STP files can be used when the FPGA has been programmed with either the POF or SOF. The SignalTap II file includes the key signals of the application logic. The `init` signal in the DMA Read and Write modules transitions to zero at the beginning of the transfer and can be used as a trigger in the SignalTap II file to capture the data. The SignalTap II file can provide information on the performance of this design. In the transmit direction, the frequent deassertion of the `tx_ws0` signal indicates that the core is not receiving enough credits from the device at the far end of the PCI Express link. In the receive direction, the deassertion of `rx_dv0` indicates that the core is not receiving enough data.
Performance
Benchmarking
Results

Tables 1 to 3 show the performance of the x8, x4, and x1 operations with the PCI Express Development Kit, Stratix II GX Edition, for two chipsets from Dell and NVIDIA using this reference design. The details of the chipset are included in the tables. The tables show the average throughput for a transfer size of 100,000 bytes and 20 iterations.

### Table 1. Stratix II GX x8 Operation

<table>
<thead>
<tr>
<th>Platform Tested [Name, Chipset, Maximum Write Payload (bytes), and Read Completion (bytes)]</th>
<th>DMA Writes (MB/s)</th>
<th>DMA Reads (MB/s)</th>
<th>Simultaneous DMA Read/Writes (MB/s) (1)</th>
<th>Theoretical Maximum Throughput (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dell 490, 5000X, 256, 64</td>
<td>1771</td>
<td>1295</td>
<td>1287/1652</td>
<td>1855/1523</td>
</tr>
<tr>
<td>NVIDIA, CK804, 128, 64</td>
<td>1632</td>
<td>1301</td>
<td>1302/1400</td>
<td>1729/1523</td>
</tr>
</tbody>
</table>

**Note to Table 1:**
(1) The simultaneous DMA read/write operation is very sensitive to the system (PC) memory controller. The measurements were obtained using four DIMMs.

### Table 2. Stratix II GX x4 Operation

<table>
<thead>
<tr>
<th>Platform Tested [Name, Chipset, Maximum Write Payload (bytes), and Read Completion (bytes)]</th>
<th>DMA Writes (MB/s)</th>
<th>DMA Reads (MB/s)</th>
<th>Simultaneous DMA Read/Writes (MB/s) (1)</th>
<th>Theoretical Maximum Throughput (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dell 490, 5000X, 256, 64</td>
<td>847</td>
<td>654</td>
<td>645/828</td>
<td>927/761</td>
</tr>
<tr>
<td>NVIDIA, CK804, 128, 64</td>
<td>819</td>
<td>654</td>
<td>652/757</td>
<td>864/761</td>
</tr>
</tbody>
</table>

**Note to Table 2:**
(1) The simultaneous DMA read/write operation is very sensitive to the system (PC) memory controller. The measurements were obtained using four DIMMs.
Table 3. Stratix II GX x1 Operation

<table>
<thead>
<tr>
<th>Platform Tested [Name, Chipset, Maximum Write Payload (bytes), and Read Completion (bytes)]</th>
<th>DMA Writes (MB/s)</th>
<th>DMA Reads (MB/s)</th>
<th>Simultaneous DMA Read/Writes (MB/s) (1)</th>
<th>Theoretical Maximum Throughput (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dell 490, 5000X, 256, 64</td>
<td>224</td>
<td>162</td>
<td>155/213</td>
<td>231/190</td>
</tr>
<tr>
<td>NVIDIA, CK804, 128, 64</td>
<td>207</td>
<td>185</td>
<td>177/199</td>
<td>216/190</td>
</tr>
</tbody>
</table>

Note to Table 3:
(1) The simultaneous DMA read/write operation is very sensitive to the system (PC) memory controller. The measurements were obtained using four DIMMs.

Table 4 shows a comparison of the throughput measured with the CATC PCI Express analyzer and the software application for the Stratix II GX device (x8 operation) and 100K bytes transfer on a Dell 490. The numbers reported by the software application and the CATC PCI Express analyzer match very closely.

Table 4. Comparison with CATC PCI Express Analyzer

<table>
<thead>
<tr>
<th>Operation</th>
<th>CATC Measurement (MB/s)</th>
<th>Altera Software Application Measurement (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA Write</td>
<td>1799</td>
<td>1771</td>
</tr>
<tr>
<td>DMA Read</td>
<td>1316</td>
<td>1295</td>
</tr>
</tbody>
</table>

Figure 15 shows the performance plateaus after a particular transfer size. The values were captured on the NVIDIA CK804 chipset using the Stratix II GX PCI Express development board operating in x8 mode.
Figure 15. Throughput and Transfer Size

![Throughput and Transfer Size Graph](image)

References

PCI Express Specifications ([www.pcisig.com](http://www.pcisig.com))

PCI Express Compiler User Guide

Stratix II GX PCI Express development kit documentation

WinDriver information ([www.jungo.com](http://www.jungo.com))

Document Revision History

Table 5 shows the revision history for this application note.

<table>
<thead>
<tr>
<th>Date and Document Version</th>
<th>Changes Made</th>
<th>Summary of Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 2007 v1.0</td>
<td>Initial release.</td>
<td>—</td>
</tr>
</tbody>
</table>