XAPP859 (v1.1) July 31, 2008

Virtex-5 FPGA Integrated Endpoint Block for PCI Express Designs: DDR2 SDRAM DMA Initiator Demonstration Platform

Authors: Kraig Lund, David Naylor, and Steve Trynosky

Summary

This application note provides a reference design for endpoint-initiated Direct Memory Access (DMA) data transfers using the LogiCORE™ Endpoint Block Plus for Virtex®-5 FPGAs. The reference design targets the ML555 hardware platform and uses the on-board DDR2 memory for storing DMA data. The design illustrates how to create an 8-lane endpoint design with an interface to a DDR2 memory. The reference design can also target an ML505 hardware platform to showcase DMA performance in a 1-lane endpoint design. A driver and GUI application are provided to allocate and initialize a system memory buffer in the host PC and set up DMA transfers. The design demonstrates how to transmit and receive transaction layer packets as a DMA initiator as well as respond to target transactions.

Introduction

The reference design in this application note provides an interface between the Virtex-5 FPGA integrated Endpoint block for PCI Express® designs and a single-rank, 64-bit, 256 MB DDR2 SDRAM memory. The reference design runs on the Virtex-5 FPGA ML555 development board for PCI Express designs. The reference design has these features:

- Supports endpoint-to-root complex DMA full duplex read and write transactions initiated by the Endpoint on the Virtex-5 FPGA ML555 development board for PCI Express designs. The host processor controls and monitors DMA transfers utilizing Programmed Input/Output (PIO) accesses to a register file contained in the endpoint FPGA design.
- Uses the LogiCORE Endpoint Block Plus wrapper for PCI Express designs, which includes the Virtex-5 FPGA integrated Endpoint block for PCI Express designs.
- Uses a DDR2 small outline dual in-line memory module (SODIMM) memory controller generated by the Xilinx® Memory Interface Generator (MIG) tool.
- Targets the XC5VLX50T-1FFG1136C production silicon FPGA.
- Requires a PC running Microsoft Windows XP with one available 8-lane PCIe add-in-card slot (the ML555 board is plugged into this slot).
- Includes a driver generated by Jungo, Ltd. WinDriver for accessing devices on the PCI bus.
- Offers user-controlled DMA initiator control and status functions through a graphical user interface (GUI) application running on the PC.

The Virtex-5 FPGA ML555 Development Kit for PCI Express designs includes the ML555 board with an 8-lane PCIe interface used to implement 4-lane or 8-lane designs. Figure 1 shows a block diagram of the system solution.

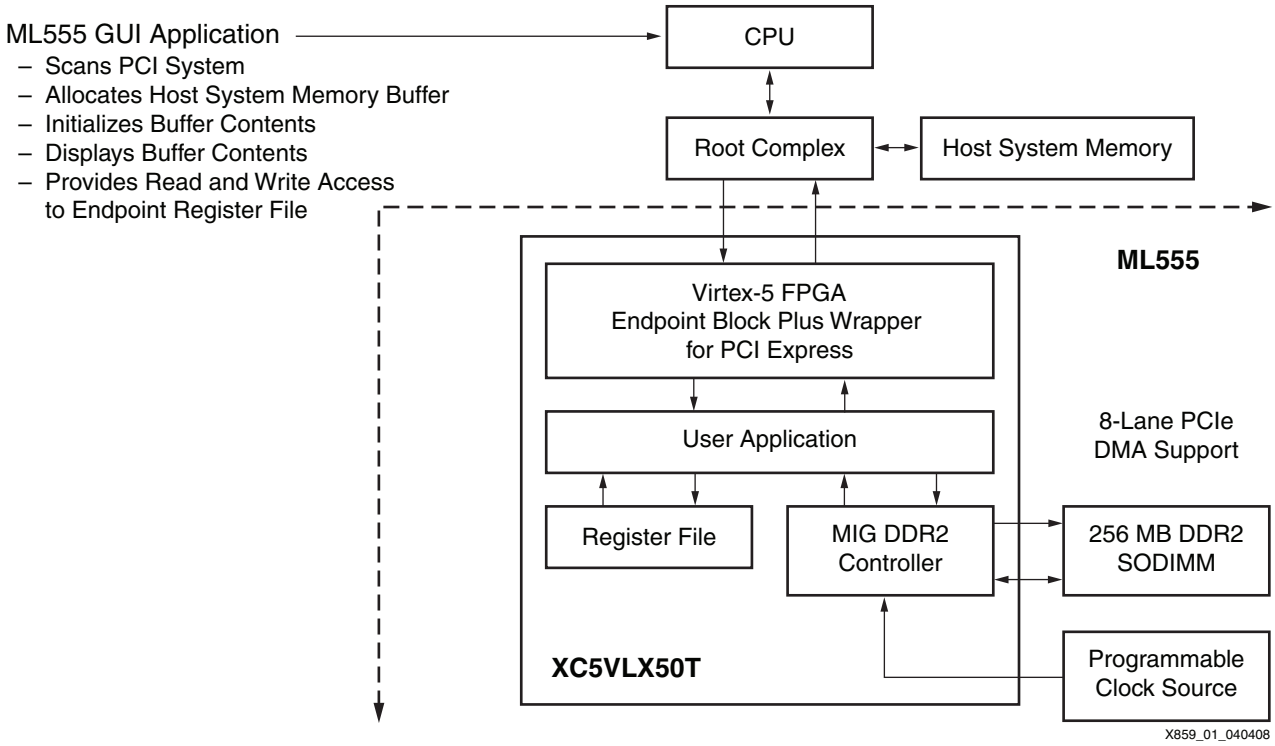


Figure 1: **System Block Diagram of the Endpoint DMA Initiator for PCI Express**

Reference System

This section describes the system design elements, including:

- Endpoint bus mastering DMA initiator control and status
- FPGA reference design elements
- Driver generated by WinDriver and user application software

Endpoint Bus Mastering DMA Initiator Control and Status

Control and status of the Endpoint DMA initiator is provided through a GUI. The GUI provides control inputs to the processor and receives status outputs from the endpoint user application. The host processor provides user control over DMA initiator functions, including:

- DMA transfer direction:
  - Read: PC host system memory transfer to ML555 DDR2 memory.
  - Write: ML555 DDR2 memory transfer to PC host system memory.
- DMA transfer size: 128, 256, 512, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, 512K, or 1M bytes. In full duplex mode, read and write DMA transfer sizes can be different. The hardware supports a DMA transfer size of 128 x m bytes (m = 1 through 8192), while the GUI application software supports 128 x 2^n bytes (n = 0 through 13).
- Number of DMA transfers: 1, 25, 50, 75, or 100. This parameter denotes the number of times to repeat a given transfer.
- Status display showing the host system memory base address: The base address is automatically determined during the ML555 GUI launch process. The system memory buffer can be initialized with a predefined pattern, and the buffer can be printed to a log window.
- Selection of host system memory address offset: 0, 128, 256, 512, 1K, 2K, 4K, 8K, 16K, 32K, or 64K bytes.

- Selection of ML555 DDR2 Endpoint memory address offset: 0, 128, 256, 512, 1K, 2K, 4K, 8K, 16K, 32K, or 64K bytes.
- Start DMA transfer options: run demonstration, read DMA, write DMA, or full duplex DMA operations.
- Display of ML555 Endpoint DMA controller register file contents.
- PCI Express configuration space display window: This window shows information relevant to the ML555 board/system PCIe link including maximum read request size, maximum payload size (MPS), read completion boundary (RCB) from the endpoint devices link control register (which can be set by the host processor's PCI Express configuration software), and negotiated link width (in lanes) of the DMA interface connection.

Demonstration mode runs through a complete sequence of read/write DMA transactions and computes DMA performance for each transfer. For a detailed explanation of transaction layer throughput, see "Understanding PCI Express Transaction Layer Throughput," page 45.

Upon completion of a DMA operation, the host PC processor firmware calculates the DMA performance for the requested transaction and prints the result to the GUI log window. The DMA performance does not include any overhead from the software setup of the host system but does include DDR2 memory access latency for DMA writes. DMA performance for reads includes the time to create memory read request packets, the turnaround time from Endpoint to root to Endpoint, and receipt of the last completion packet with data. DMA read performance does not include the time to write data into the ML555 DDR2 memory.

FPGA Reference Design Elements

This section describes the functions of the main FPGA design elements included in the reference design. The top-level design file is called `pcie_dma_top` and instantiates several low-level modules:

- The LogiCORE Endpoint Block Plus wrapper for PCI Express designs found in the CORE Generator™ software is titled `endpoint_blk_plus_<version number>`.
- The user application logic wrapper for PCI Express designs is titled `pcie_dma_wrapper`.
- The user application logic to DDR2 controller wrapper for PCI Express designs is titled `dma_ddr2_if`.
- The MIG DDR2 memory controller top-level design file is titled `mem_interface_top`.

LogiCORE Endpoint Block Plus Wrapper for PCI Express Designs

Access to the LogiCORE Endpoint Block Plus wrapper is delivered in simulation-only mode to allow evaluation of the core in a system-level simulation. ISE® software, version 10.1 or higher must be installed on the system. The license to use the Endpoint Block Plus wrapper for PCI Express designs is provided at no-charge. To register, the designer is required to accept the licensing agreement and must be registered to gain access to the protected area lounge. Users need to register and obtain a full license to generate a bitstream. The license request area is password protected, but all licensed Xilinx customers can request access to this area located on the Xilinx website at:
http://www.xilinx.com/ipcenter/pcie_blkplus_lounge/pcie_blkplus_registration.htm.

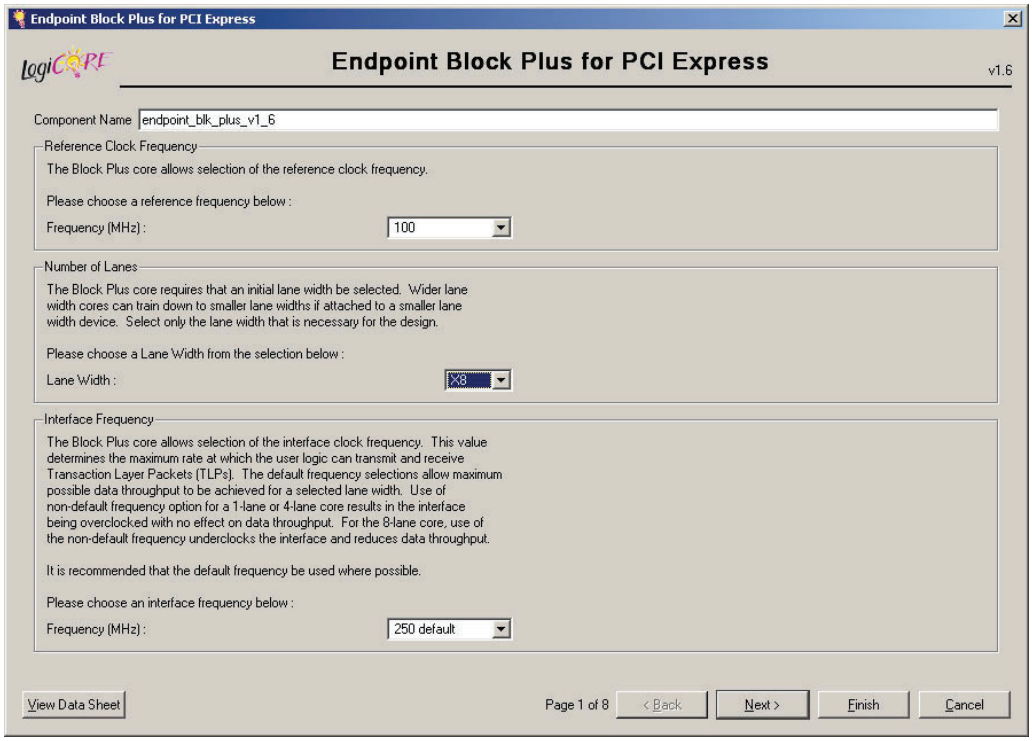
For this design, the CORE Generator software is used to create the Virtex-5 FPGA Endpoint Block Plus wrapper for PCI Express. The wrapper configures the integrated transceivers and integrated Endpoint block. It also connects the GTP transceiver to the integrated Endpoint block, and connects block RAM elements for the transmit, receive, and retry buffers.

For additional information on this LogiCORE product, please go to the following sites:

- http://www.xilinx.com/products/ipcenter/V5_PCI_Express_Block_Plus.htm
- http://www.xilinx.com/support/documentation/ipbusinterfacei-o_pci-express_v5pciexpressblockplus.htm

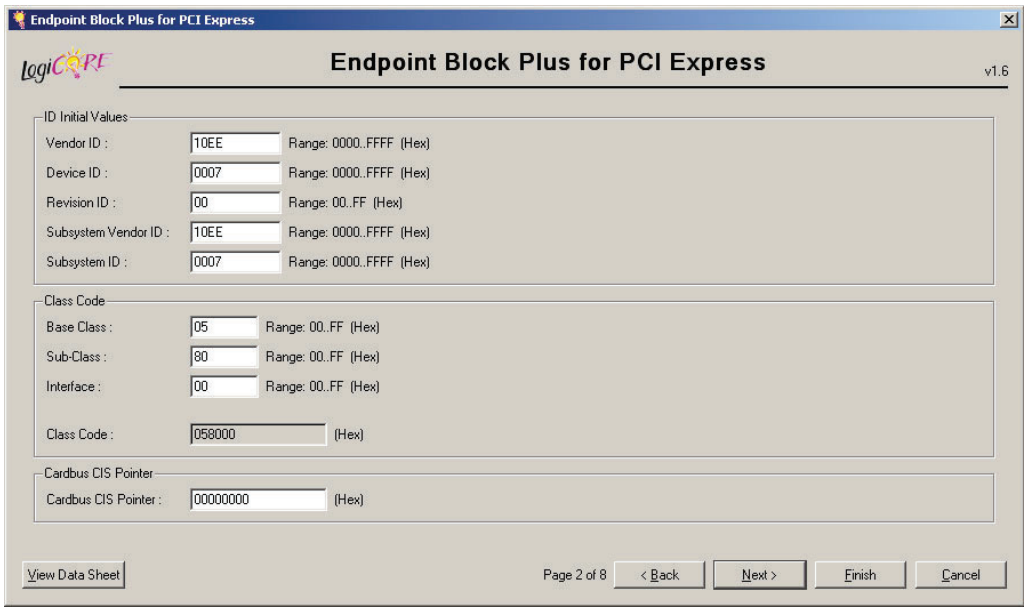
- http://www.xilinx.com/support/documentation/ip_documentation/xtp025.pdf

The screen captures in Figure 2 to Figure 9 from the CORE Generator software show the parameters created using the Endpoint Block Plus for PCI Express in this reference design.



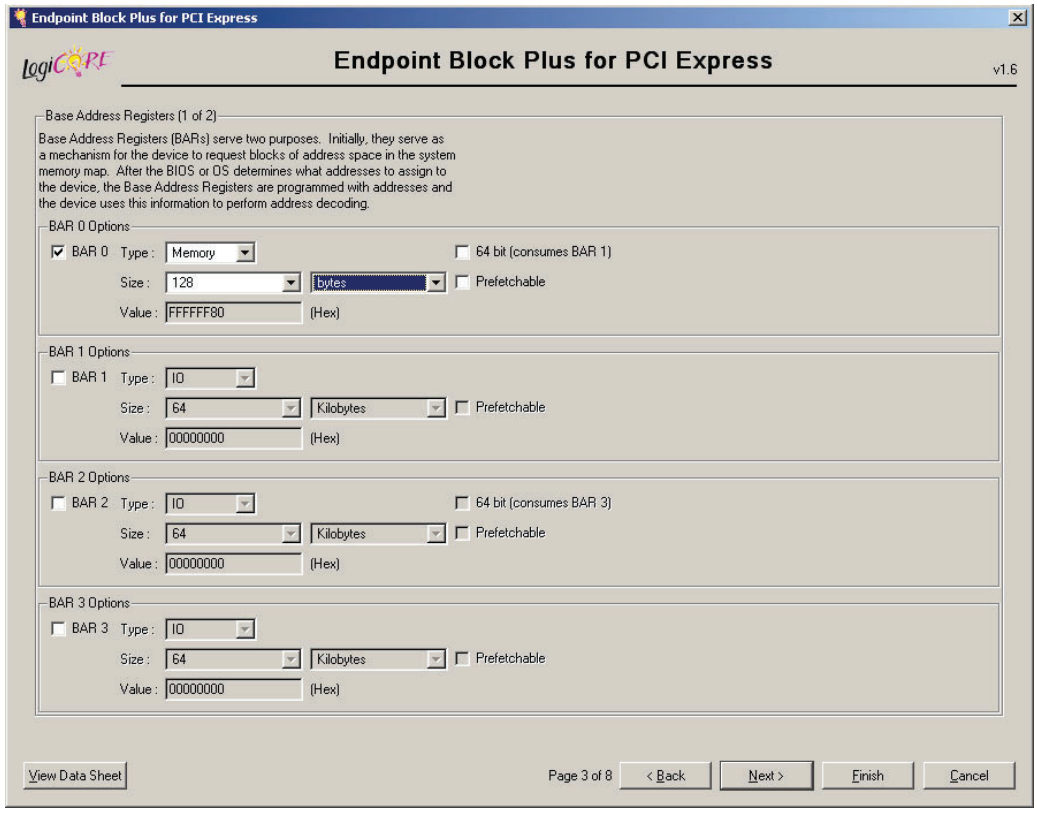
X859_02_041008

Figure 2: **Endpoint Block Plus Wrapper for PCI Express, Page 1 of 8**



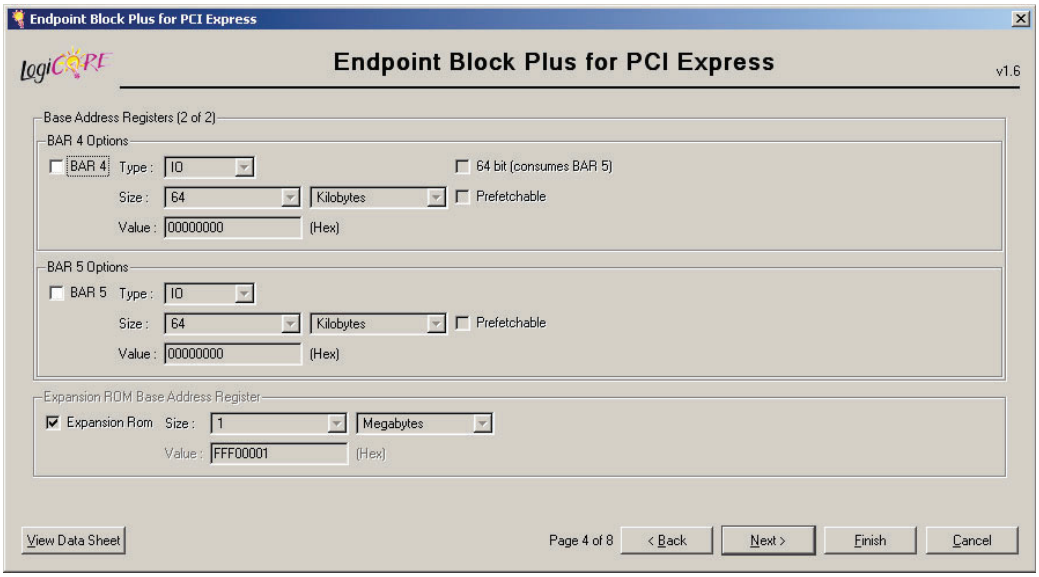
X859_03_041008

Figure 3: **Endpoint Block Plus Wrapper for PCI Express, Page 2 of 8**



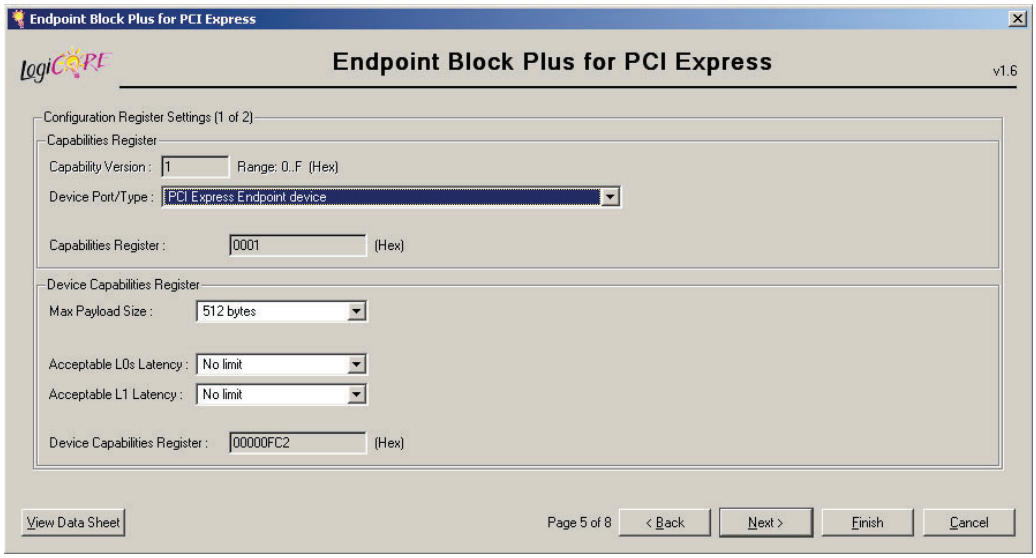
X859_04_041008

Figure 4: **Endpoint Block Plus Wrapper for PCI Express, Page 3 of 8**



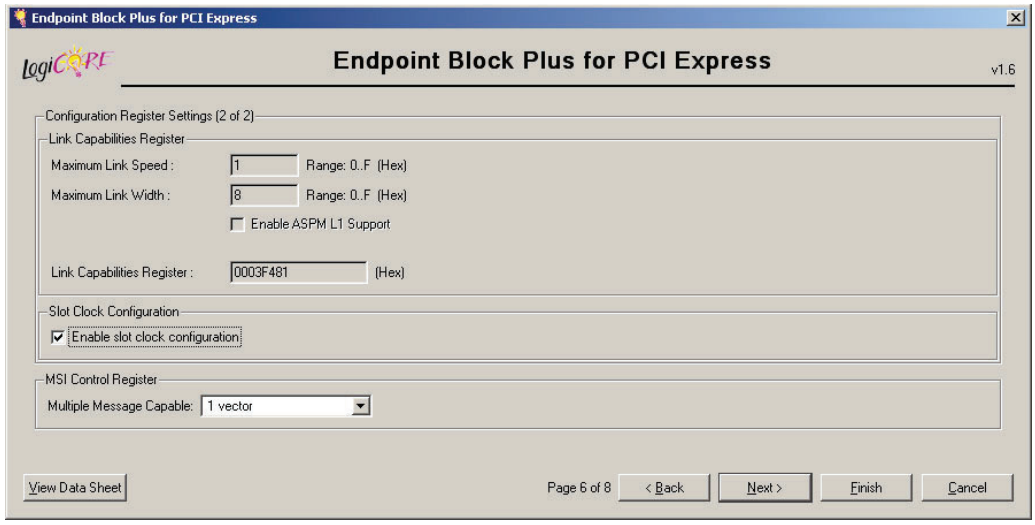
X859_05_041008

Figure 5: **Endpoint Block Plus Wrapper for PCI Express, Page 4 of 8**



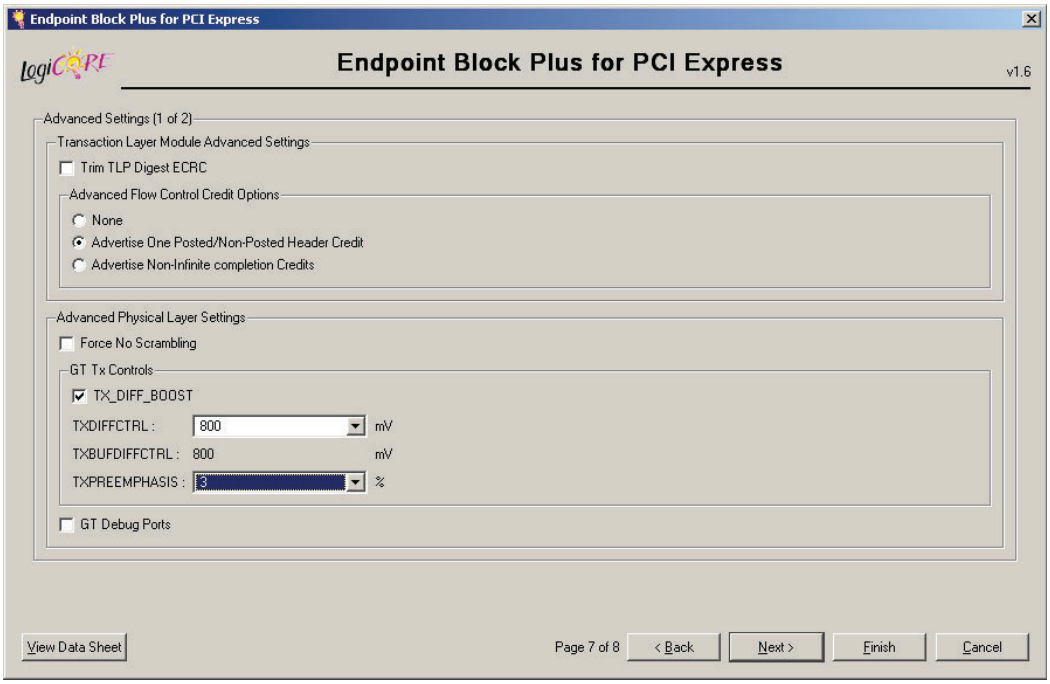
X859_06_041008

Figure 6: **Endpoint Block Plus Wrapper for PCI Express, Page 5 of 8**



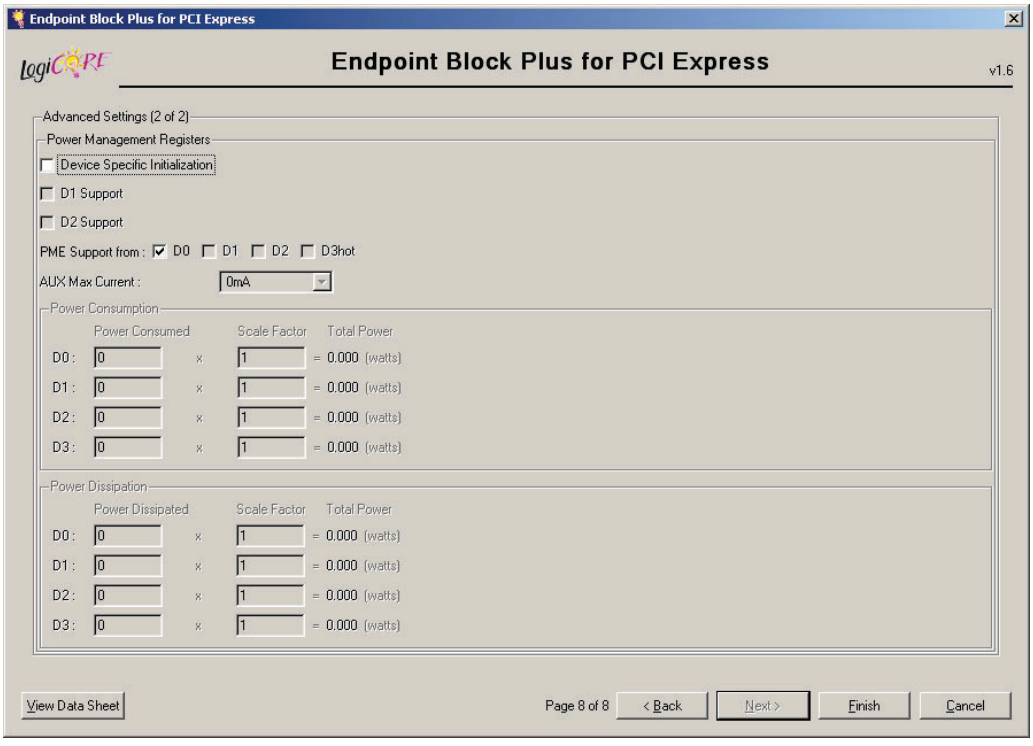
X859_07_041008

Figure 7: **Endpoint Block Plus Wrapper for PCI Express, Page 6 of 8**



X859_08_040908

Figure 8: **Endpoint Block Plus Wrapper for PCI Express, Page 7 of 8**



X859_09_041008

Figure 9: **Endpoint Block Plus Wrapper for PCI Express, Page 8 of 8**

The *Endpoint Block Plus for PCI Express Product Specification* [Ref 1], the *LogiCORE Endpoint Block Plus for PCI Express User Guide* [Ref 2], and the *LogiCORE Endpoint Block Plus for PCI Express Getting Started Guide* [Ref 3] are essential in explaining the design, simulation, and user interface of the Endpoint Block Plus wrapper.

Note: The Endpoint Block Plus for PCI Express documentation referenced above includes the core version number. The latest document revisions are available at http://www.xilinx.com/products/ipcenter/V5_PCI_Express_Block_Plus.htm.

User Application Logic

The user application logic provides the interface between the Endpoint Plus Block wrapper for PCI Express, the DMA Control/Status Register File, and the DDR2 SODIMM interface. Six major blocks are included in this interface, as shown in Figure 10:

- TX Engine
- RX Engine
- DMA to DDR2 Interface
- Egress Data Presenter
- Read Request Wrapper
- DMA Control/Status Registers

Two clock domains are used in this design. See "Clocks," page 25, for additional information.

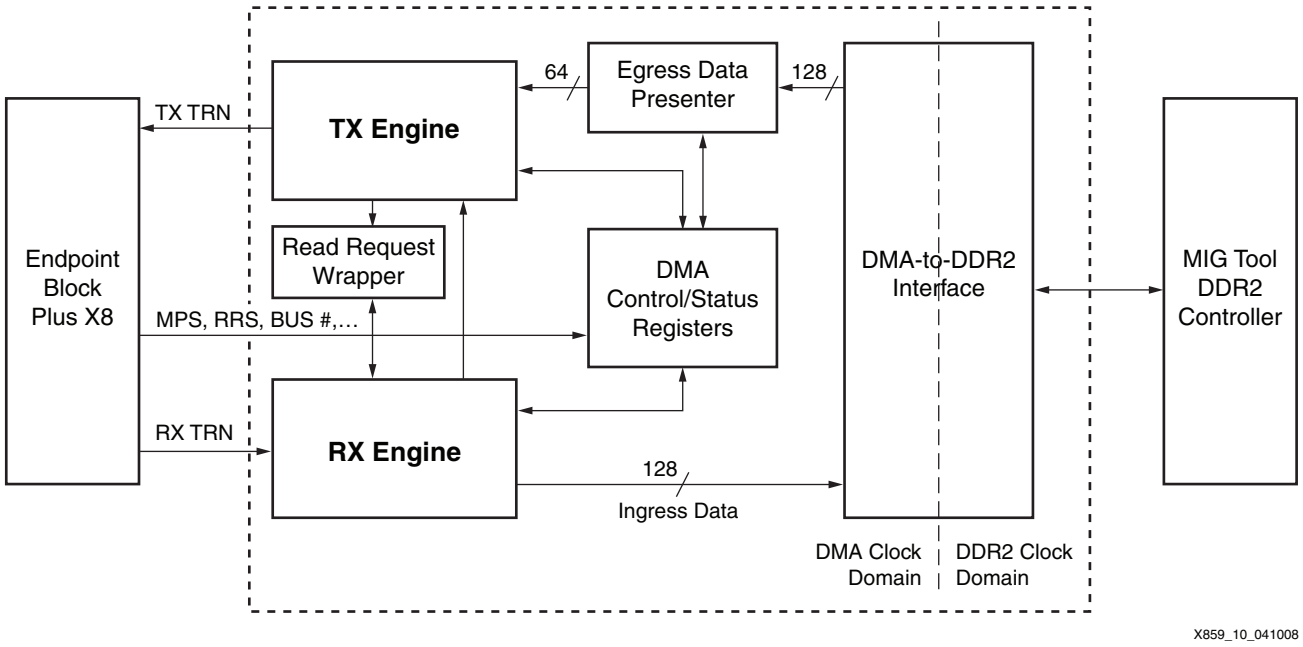


Figure 10: **User Application Diagram**

TX Engine

The TX engine creates and transmits posted, non-posted, and completion packets. The reference design can generate and transmit posted memory write, non-posted memory read, and completion packets to satisfy memory read and write DMA requests. Message or error reporting packets, such as Unsupported Request, Completion Timeout, and Interrupt Request, are generated by the Endpoint Block Plus wrapper through a simple signaling mechanism. Therefore, it is not necessary for the user logic to create the packet headers for these types of transaction layer packets (TLPs); the user logic only needs to monitor for the conditions that warrant them and signal to the Endpoint Block Plus when such conditions occur. Completion timeout logic is not implemented in this reference design. Additionally, I/O packets are not supported by this reference design. Packets are created upon request from the DMA Control/Status Register File interface. They are transmitted to the Endpoint Block Plus wrapper through the transmit (TX) transaction (TRN) interface.

Figure 11 shows a block diagram of the TX engine.

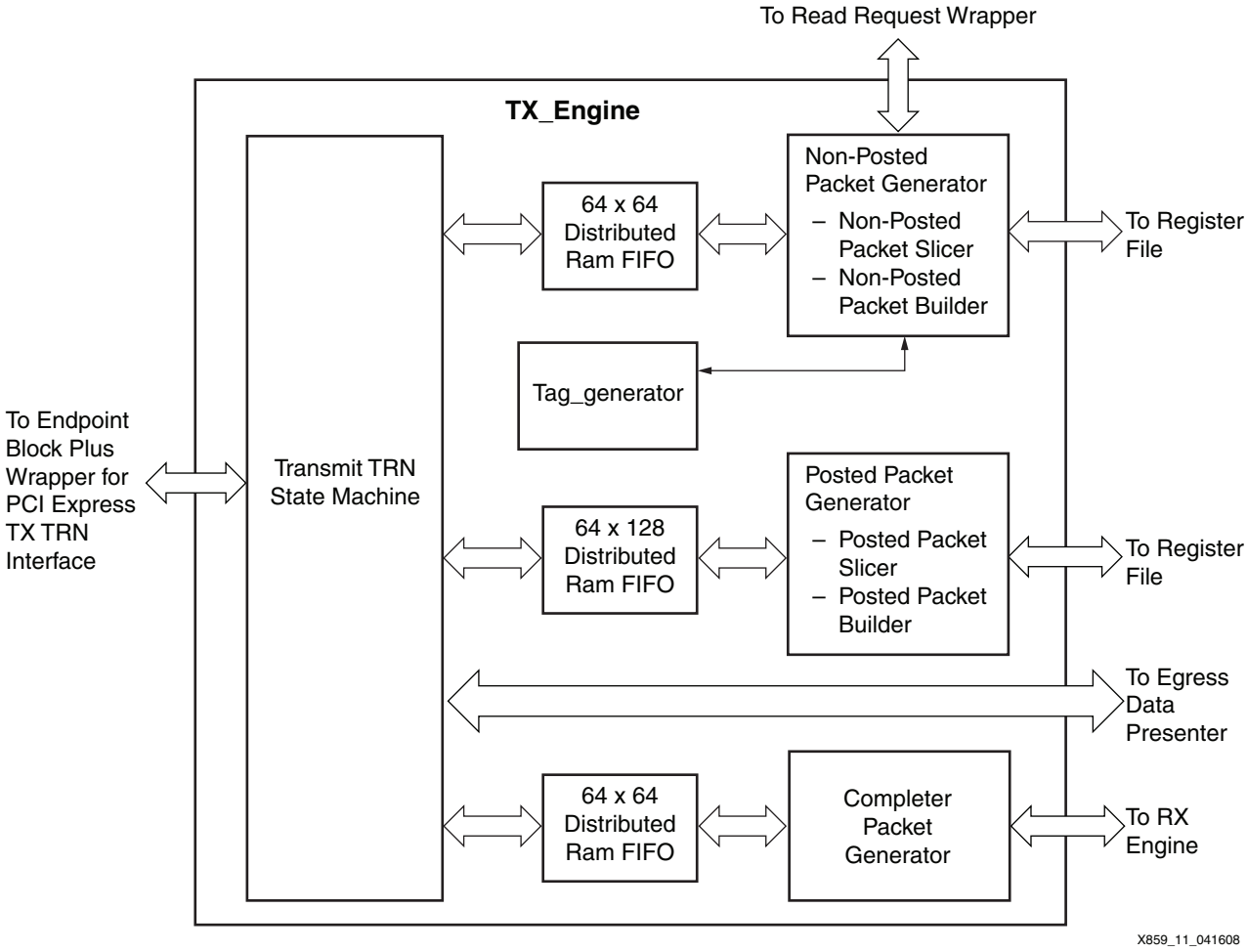


Figure 11: **TX Engine Block Diagram**

Posted Packet Generator

The posted packet generator accepts write DMA requests from the DMA Control/Status Register File block and generates the posted-packet headers required to complete a write DMA transaction. These headers are placed into a small FIFO to be read by the TX TRN state machine block. Two main logical blocks make up the posted packet generator:

- Posted Packet Slicer
- Posted Packet Builder

Posted Packet Slicer Block

The posted packet slicer accepts write DMA requests from the DMA Control/Status Register File block and "slices" these requests into multiple PCI Express packets based on the rules set forth in the PCI Express Base Specification [Ref 4]. Essentially, the posted packet slicer breaks up write DMA requests based on two rules:

- Packet data payload length does not exceed the Max_Payload_Size field of the device control register.
- Address/Length combinations cannot cross a 4 KB address boundary.

The posted packet slicer block passes the length (length[9:0]) and destination address (dmawad_reg[63:0]) fields to the posted packet builder block using a simple handshaking

protocol (go/ack signals). Source and destination addresses must be aligned on 128-byte address boundaries. The posted packet slicer block is designed to take advantage of all possible Max_Payload_Size values and can accept transfer size inputs (dmawxs[31:0]) of 128, 256, 512, 1024, 2048, and 4096 bytes.

Posted Packet Builder Block

The posted packet builder block builds posted memory write request packet headers, and then writes the headers into a small, 64 x 128, CORE Generator software FIFO built with distributed RAM. This block is capable of dynamically generating 3DW or 4DW headers based on the destination address (dmawad[63:0]) input. If dmawad[63:32] is zero, a 3DW header is built; otherwise, a 4DW header is built. Figure 12 shows a 3DW and 4DW posted memory write packet header.

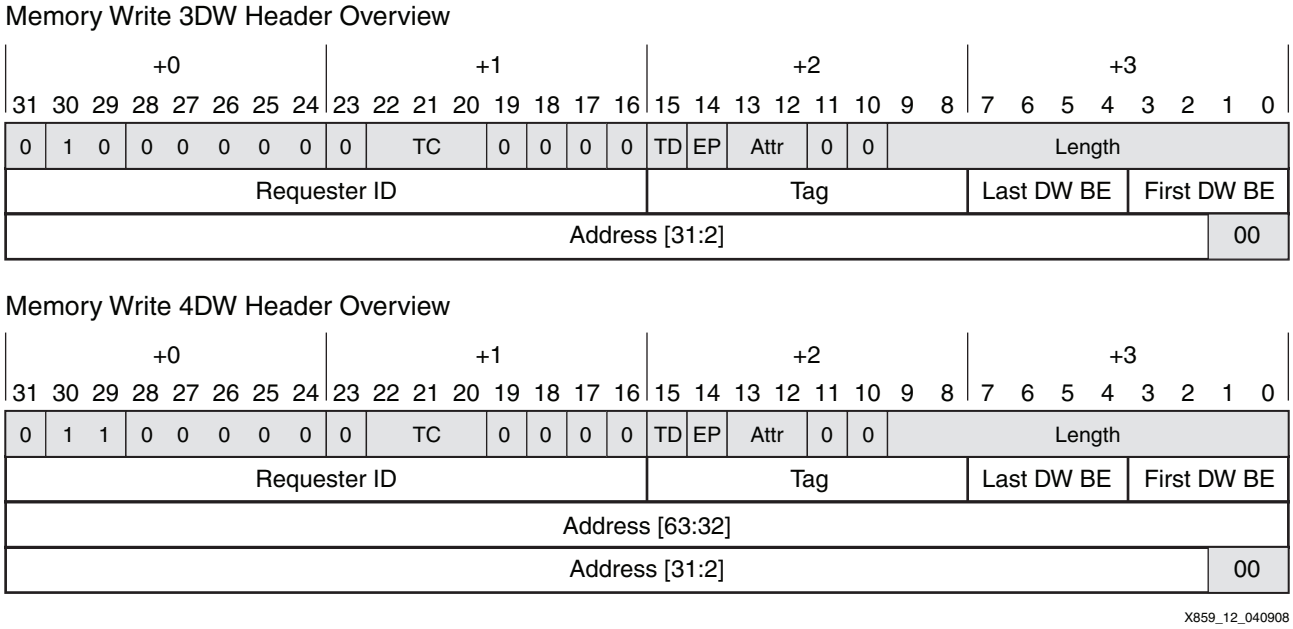


Figure 12: **3DW and 4DW Posted Memory Write Packet Headers**

Table 1 describes the posted memory write header fields.

Table 1: **Posted Memory Write Header Field Information**

Header Field	Description
TC[2:0]	Traffic class field. Always `000` for this design.
TD	TLP Digest bit. Always `0`. The design does not implement end-to-end cyclic redundancy check (ECRC).
EP	Poisoned bit. Always `0`. Not used in this design.
ATTR[1:0]	Attributes field. Always `00` for memory writes.
Length[9:0]	Length of data payload in DWORDS. This field is provided by the posted packet slicer block.
Requester ID[15:0]	Requester ID field. This is provided by the Endpoint Block Plus wrapper for PCI Express.
Tag[7:0]	Tag field. This reference design always sets the Tag field to `00000000` for all posted packets.
LastBE[3:0]	Last DWORD byte enable field. Always `1111` for this design because memory writes are aligned on 128-byte address boundaries.

Table 1: **Posted Memory Write Header Field Information *(Cont'd)***

Header Field	Description
FirstBE[3:0]	First DWORD byte enable field. Always `1111` for this design because memory writes are always more than one DWORD in length.
Address[63:32]	Upper 32-bit host memory address for 64-bit systems using memory above 4 GB. If the host processor writes a non-zero value to the DMAWAD_U register in the DMA Control/Status Register File (Table 3), the design uses 64-bit addressing and 4DW TLP packet headers.
Address[31:0]	Lower 32-bit host memory address for 32-bit systems or 64-bit systems using memory below 4 GB.

Non-Posted Packet Generator

The non-posted packet generator accepts read DMA requests from the DMA Control/Status Register File block and generates the non-posted packet headers required to complete a read DMA transaction. These headers are placed into a small FIFO to be read by the TX TRN state machine block. Two main logical blocks make up the non-posted packet generator:

- Non-posted packet slicer
- Non-posted packet builder

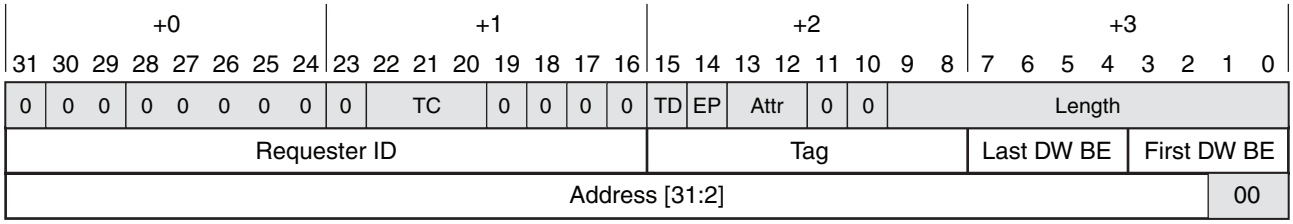
The non-posted packet slicer block accepts read DMA requests from the DMA Control/Status Register File block and "slices" these requests into multiple PCI Express packets based on the rules set forth in the PCI Express Base Specification [Ref 4]. Essentially, the non-posted packet slicer breaks up read DMA requests based on these rules:

- Requested data payload length does not exceed the Max_Read_Request_Size field of the device control register.
- Address/length combinations cannot cross a 4 KB address boundary.

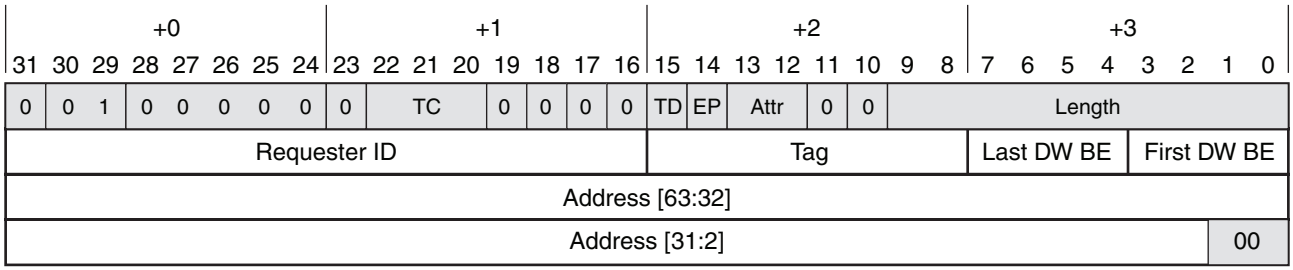
The non-posted packet generator block passes the length (length[9:0]) and source address (dmaras_reg[63:0]) fields to the non-posted packet builder block, using a simple handshaking protocol (go/ack signals). Source and destination addresses must be aligned on 128-byte address boundaries. The non-posted packet generator block is designed to take advantage of all possible Max_Read_Request_Size values and can accept transfer size inputs (dmarxs[31:0]) of 128, 256, 512, 1024, 2048, and 4096 bytes.

The non-posted packet builder block builds non-posted memory read request packet headers, and then writes the headers into a small 64 x 64 CORE Generator software FIFO built with distributed RAM. Also, this block is capable of dynamically generating 3DW or 4DW headers based on the source address (dmaras[63:0]) input. If dmaras[63:32] is zero, a 3DW header is built; otherwise, a 4DW header is built. Figure 13 shows a 3DW and 4DW posted memory read packet header.

Memory Read 3DW Header Overview



Memory Read 4DW Header Overview



X859_13_040908

Figure 13: **3DW and 4DW Non-Posted Memory Read Packet Headers**

Table 2 describes the non-posted memory read header fields.

Table 2: **Non-Posted Memory Read Header Field Information**

Header Field	Description
TC[2:0]	Traffic class field. Always `000` for this design.
TD	TLP digest bit. Always `0`. Design does not implement ECRC.
EP	Poisoned bit. Always `0`. Not used in this design.
ATTR[1:0]	Attributes field. Always `10` for memory reads. The relaxed ordering bit must be set to comply with the requirements of using the completion streaming mode. Refer to the *LogiCORE Endpoint Block Plus for PCI Express User Guide* [Ref 2] for more details.
Length[9:0]	Length of data payload in DWORDS. This field is provided by the non-posted packet slicer block.
Requester ID[15:0]	Requester ID field. This is provided by the Endpoint Block Plus wrapper for PCI Express.
Tag[7:0]	Tag field. A unique tag provided by tag generator block.
LastBE[3:0]	Last DWORD byte enable field. Always `1111` for this design because memory reads are aligned on 128-byte address boundaries.
FirstBE[3:0]	First DWORD byte enable field. Always `1111` for this design because memory reads are always more than one DWORD in length.
Address[63:32]	Upper 32-bit host memory address for 64-bit systems using memory above 4 GB. If the host processor writes a non-zero value to the DMARAS_U register in the DMA Control/Status Register File (Table 3), the design uses 64 bit addressing and 4DW TLP packet headers.
Address[31:0]	Lower 32-bit host memory address for 32-bit systems or 64-bit systems using memory below 4 GB.

The non-posted packet builder block also provides some information to the RX engine so that the RX engine can qualify the resulting incoming completion packets from the non-posted memory read request packets and place the data payload in the correct DDR2 memory

address. This information is provided to the RX engine through a small buffer called the read request RAM. For more information on this FIFO, refer to "Read Request Wrapper," page 20.

Completer Packet Generator

The completer packet generator accepts memory read requests from the RX engine and generates completer packet headers. These headers are placed into a small FIFO to be read by the TX TRN state machine block.

Tag Generator

The tag generator block generates unique tags upon request from the posted (not required) and non-posted (required) packet generator blocks. The tags are generated from a simple 5-bit counter that rolls over to zero upon reaching its maximum value. The extended tag function is not supported by this reference design. Simple arbitration logic grants access to the tag counter between the posted and non-posted packet generator blocks.

Transmit TRN State Machine

The transmit TRN state machine block sends packets to the TX TRN interface of the Endpoint Block Plus wrapper. The transmit TRN state machine block loads header information from packet generator FIFOs. The 64-bit data is sourced from the egress data presenter block for posted memory writes (and completer packets). The state machine and datapath pipeline are clock enabled by the trn_tdst_rdy_n signal from the Endpoint Plus block wrapper. When deasserted, this signal signifies that the Endpoint Plus block wrapper is not ready to receive packet headers or data, and must immediately stop transmission to the TX TRN interface. The state machine and datapath are also clock enabled by a data_stall signal from the DMA-to-DDR2 interface block. This signal is asserted if the DMA-to-DDR2 interface temporarily runs out of data provided to the egress data presenter block, which is possible if a DDR2 memory refresh cycle occurs in the middle of data transmission. The transmit TRN state machine block is also responsible for signaling to the DMA Control/Status Register File when a write DMA transaction is complete.

RX Engine

The RX engine receives packets from the Endpoint Block Plus wrapper RX TRN interface and routes the data payload to the correct memory address. The RX engine is informed of pending read requests from the TX engine through the read request wrapper. Supported packet types are completions, memory reads, and memory writes. This reference design does not provide an I/O BAR function, so I/O packets are filtered by the Endpoint Block Plus wrapper. Message packets are filtered by the RX engine. Figure 14 shows a block diagram of the RX engine.

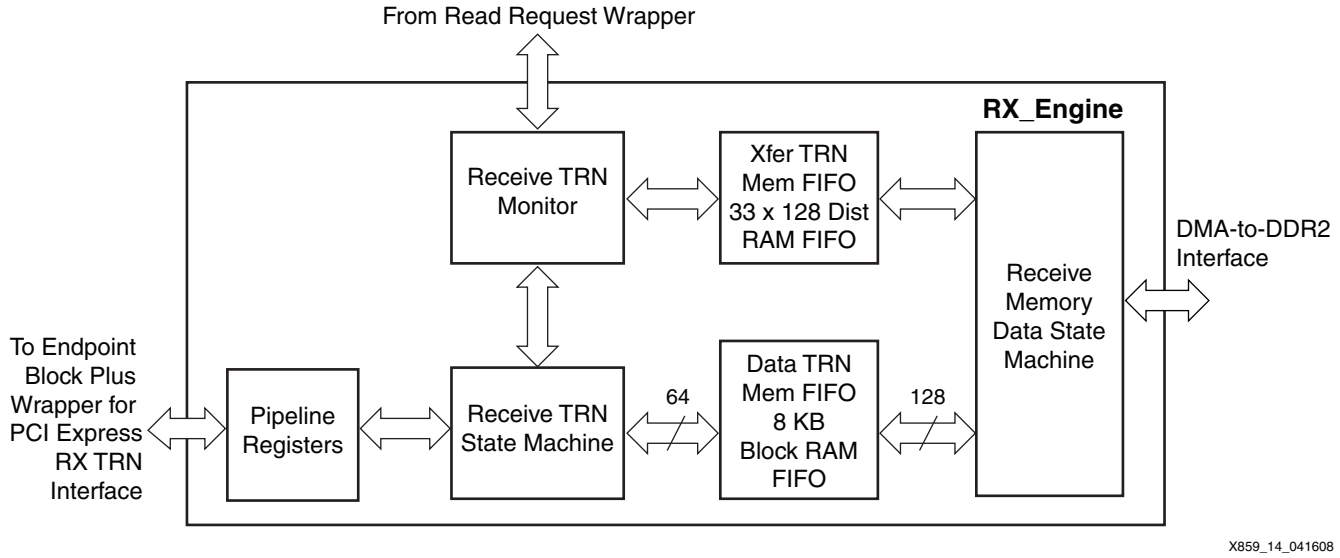


Figure 14: **RX Engine Block Diagram**

Receive TRN State Machine

The receive (RX) TRN state machine block performs these functions:

- Drives the RX TRN interface inputs on the Endpoint Block Plus wrapper for PCI Express, and throttles the interface if necessary
- Presents 64-bit data along with byte enables
- Drives a data_valid signal to qualify the data and byte enables
- Decodes, registers, and qualifies packet header information for the receive TRN monitor block

In this implementation, the 64-bit datapath is connected directly to a memory buffer with the data_valid signal being used as a write enable. This is done because all completer packets are currently routed to the DDR2 memory. In cases where the data can have multiple destinations, additional decoding is necessary in this block to steer the data. For example, in a scatter-gather or chaining DMA design, completer packets need to be steered to a different memory. Also, the bar_hit signals from the Virtex-5 FPGA integrated Endpoint block for PCI Express can facilitate data routing for posted memory writes.

Receive TRN Monitor

The receive TRN monitor block performs these functions:

- Reads the read request RAM and detects when a read request has been fully completed (typically a multiple of completion packets with 64-byte payloads)
- Uses the destination address information in the read request RAM along with the length field in the completer packet headers to calculate the DDR2 destination addresses
- Calculates when a read DMA transaction has been completed and signals to the DMA Control/Status Register File
- Provides a byte address and transfer size to the receive memory data finite state machine (FSM) block for each fully and successfully received completer

Data TRN Memory FIFO and Xfer TRN Memory FIFO

The data TRN memory FIFO and Xfer (Transfer) TRN memory FIFO operate as a pair. The data TRN memory FIFO has two functions. First, it buffers incoming data from received completion packets. Second, it converts the datapath from 64 bits to 128 bits for the DDR2

controller, which is based on the MIG tool. The Xfer TRN memory FIFO holds the starting address and transfer size information for the receive memory data state machine block. Each entry in the Xfer TRN memory FIFO must have the corresponding data located in the data TRN memory FIFO.

Receive Memory Data FSM

The receive memory data FSM has two main tasks:

- Arbitrates for control of the DMA-to-DDR2 interface block
- Reads data and transfers information from the data and Xfer TRN memory FIFOs and forwards it to the DMA-to-DDR2 interface block

DMA-to-DDR2 Interface

The DMA-to-DDR2 interface block is the main interface block between the PCIe DMA user application logic and the MIG DDR2 controller logic. It provides several functions for this reference design:

- Clock domain change between the DMA clock domain and MIG DDR2 clock domain
- Arbitration for ingress and egress datapaths
- MIG DDR2 Address and Command generation

Requests to access the DDR2 memory are received from the user application logic and presented to the top-level DDR2 controller for read or write access to the Endpoint memory. DDR2 controller commands are formatted as specified in *High Performance DDR2 SDRAM Interface in Virtex-5 Devices* [Ref 5]. This logic element provides these functions:

- For memory read DMA operations, DMA-to-DDR2 interface block receives data from the system memory, transfer size, start address for the DDR2 memory write, and a data request signal. The controller responds with a data acknowledge.
- For memory write DMA operations, interface block receives a transfer size, start address for the DDR2 memory read, and a data request signal. The controller responds with data to be transmitted to the system memory and a data acknowledge signal.

DMA Control and Status Wrapper

The DMA control and status wrapper is the main block with which the host system processor communicates and the main control for DMA operations. The wrapper is composed of two logical blocks:

- DMA Control/Status Register File
- Internal Control Block

Figure 15 shows the high-level block diagram overview of the wrapper and the connections between the two subblocks.

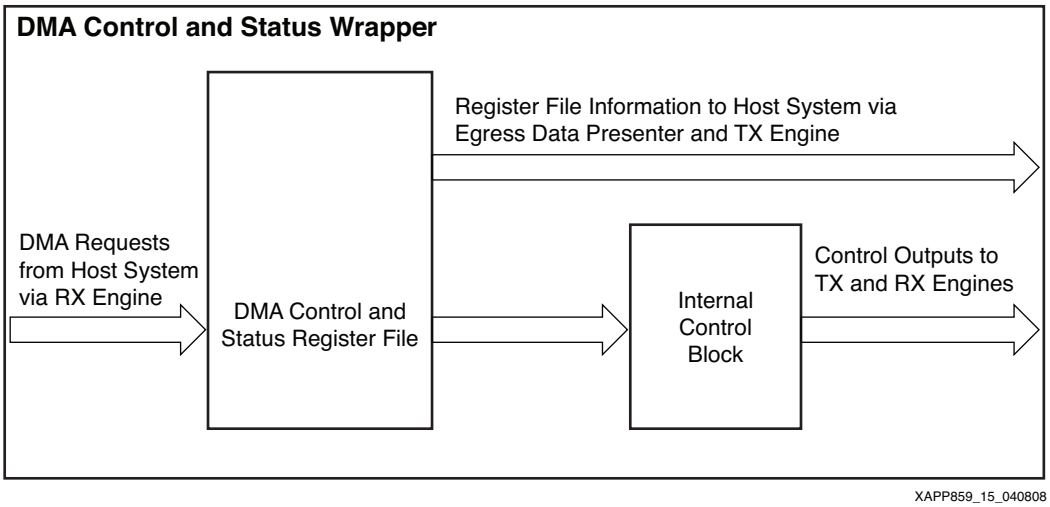


Figure 15: **Functional Overview of DMA Control and Status Wrapper**

The logical blocks in this reference design are designed to operate on DMA transfers up to a maximum of 4 KB at a time. This includes the posted packet slicer, non-posted packet slicer, and DMA-to-DDR2 interface blocks. To support transfer sizes larger than 4 KB via the hardware, it is necessary to break up large DMA transfers, i.e., transfers larger than 4 KB, into smaller chunks that the overall hardware design can handle. The DMA Control/Status Register File and Internal Control Block work in conjunction to allow transfer sizes much larger than 4 KB.

The DMA Control/Status Register File provides these general functions:

- Provides a memory-mapped interface to the host system in the form of a simple register file interface using one base address register (BAR)
- Accepts read and write DMA transfer requests up to 1 MB in size
- Breaks up DMA transfer requests larger than 4 KB into a sequence of smaller DMA transfers and feeds these small transfers to the Internal Control Block

The Internal Control Block provides these general functions:

- Accepts read and write DMA transfer requests up to 4 KB in size from the DMA Control/ Status Register File block
- Provides the main control interface to the RX and TX engine blocks

The DMA Control/Status Register File and the Internal Control Block are described in more detail in the following sections.

DMA Control/Status Register File

The DMA Control/Status Register File is memory mapped into the PCI memory space and connected to the user application logic. The host processor accesses the register file through memory read and write cycles on the bus. An Endpoint Block Plus memory base address register facilitates access to the DMA Control/Status Register File.

DMA operations are defined in the DMA Control/Status Register File. The processor initiates a DMA transfer by initializing the control registers. DMA completion is provided to the processor through status registers. Table 3 contains a description of the register file. All registers are 32-bit read/write access registers, except as noted.

Table 3: **DMA Control/Status Register File Definition**

Register Name	Abbreviation	Offset (1)	Description
DMA Write: Endpoint DDR2 Address Source	DMAWAS	x00	Endpoint DDR2 memory address for DMA write operation. The ML555 board provides 256 MB of memory.
DMA Write: System Memory Address Destination Lower	DMAWAD_L	x04	Lower 32-bit address of system memory buffer for DMA write operation. Used in 32-bit and 64-bit address routing systems.
DMA Write: System Memory Address Destination Upper	DMAWAD_U	x08	Upper 32-bit address of system memory buffer for DMA write operation. Used in 64-bit address routing systems. A non-zero value in this register causes the design to use 64-bit addressing and 4DW TLP headers, as shown in Figure 12.
DMA Read: System Memory Address Source Lower	DMARAS_L	x0C	Lower 32-bit address of system memory buffer for DMA read operation. Used in 32-bit and 64-bit address routing systems.
DMA Read: System Memory Address Source Upper	DMARAS_U	x10	Upper 32-bit address of system memory buffer for DMA read operation. Used in 64-bit address routing systems. A non-zero value in this register causes the design to use 64-bit addressing and 4DW TLP packet headers, as shown in Figure 13.
DMA Read: Endpoint DDR2 Address Destination	DMARAD	x14	Endpoint DDR2 memory address for DMA read operation. The ML555 board provides 256 MB of memory.
DMA Write Transfer Size	DMAWXS	x18	Number of bytes to transfer from Endpoint memory to system memory. Supported transfer size options are defined by the equation: 128 x 2^n (n = 0 through 13). The hardware supports transfer sizes of 128 x m bytes with m = 1 through 8192. The GUI application software supports DMA transfer sizes of 128 x 2^n (n = 0 through 13).
DMA Read Transfer Size	DMARXS	x1C	Number of bytes to transfer from system memory to Endpoint memory. Supported transfer size options are defined by the equation: 128 x 2^n (n = 0 through 13). The hardware supports transfer sizes of 128 x m bytes with m = 1 through 8192. The GUI application software supports DMA transfer sizes of 128 x 2^n (n = 0 through 13).
Reserved		x20	Reserved for future use.

Table 3: **DMA Control/Status Register File Definition *(Cont'd)***

Register Name	Abbreviation	Offset (1)	Description
Reserved		x24	Reserved for future use.
DMA Control/Status	DMACST	x28	DMA control and status registers. The DMA control register is a write-only register. The DMA status register is a read-only register. Table 4 contains a description of individual control and status bits.
Reserved		x2C	Reserved for future use.
DMA Write Transfer Counter (2)	DMAWRP	x30	This 32-bit read-only counter provides a mechanism to measure the DMA write performance of a transaction. The counter starts when the Write DMA Start bit is asserted, and ends when the end of frame is asserted on the last write packet. The counter does not reset. Performance calculations need to read the counter value before starting a transaction and compute the difference between the start and end values.
DMA Read Transfer Counter (2)	DMARDP	x34	This 32-bit read-only counter provides a mechanism to measure the DMA read performance of a transaction. The counter starts when the Read DMA Start bit is asserted, and ends when the end of frame is asserted on the last completer-with-data packet received from the host side of the PCIe interface. The counter does not reset. Performance calculations need to read the counter value before starting a transaction and compute the difference between the start and end values.

Notes:

1. The base address of the register file is defined in the Endpoint Block Plus memory BAR0.
2. These status registers are read only. All other registers provide read and write access by the processor.

The read direction denotes ingress to the DDR2 memory. The write direction denotes egress from the Endpoint DDR2 memory. Source and destination addresses must be on 128-byte boundaries. The DMA control and status registers (DMACST) provide control and status of the requested DMA operation as shown in Table 4.

Table 4: **DMA Control and Status (DMACST) Register Bit Definitions**

Field	Bit Positions	Initial Value	Description
Write DMA Start	0	0	Start write DMA operation. The control firmware writes a logic 1 to start the requested DMA operation. This bit is self clearing by the hardware.
Write DMA Done	1	0	Write DMA operation complete. The control firmware must write a logic 1 to clear this status bit.
Read DMA Start	2	0	Start read DMA operation. The control firmware writes a logic 1 to start the requested DMA operation. This bit is self clearing by the hardware.
Read DMA Done	3	0	Read DMA operation complete. The control firmware must write a logic 1 to clear this status bit.
Reserved	31:5	0	Reserved bits are defined as read only and return a value of 0. All other defined bits can be read and written by the processor firmware.
Memory PHY Initialization Complete	4	0	Memory physical initialization complete. Logic 1 indicates completion of the initialization and calibration of the DDR2 memory interface. Commands and write data cannot be written to the memory controller user interface until calibration is complete. See *Xilinx Memory Interface Generator (MIG) User Guide* [Ref 7] for additional information on memory initialization. The GUI application verifies that memory initialization is complete prior to initiating a DMA operation.

The host PC processor firmware performs these register file accesses to perform a DMA Write operation:

- Verifies that the Write DMA Start bit in DMACST[0] is `0`.
- Writes appropriate values to the DMAWAS, DMAWAD_L, DMAWAD_U, and DMAWXS registers for the user-requested DMA transfer specified in the ML555 GUI.
- Writes a `1` to DMACST[0] to initiate the DMA write operation.
- Polls the Write DMA Done bit to determine when the DMA operation is complete.
- Computes the DMA write performance based upon the user-requested transfer size and read-only counter values in the register file. Outputs the result to the GUI log window.

Host PC processor firmware performs these register file accesses to perform a DMA Read operation:

- Verifies that the Read DMA Start bit in DMACST[2] is `0`.
- Writes appropriate values to the DMARAS_L, DMARAS_U, DMARAD, and DMARXS registers for the user-requested DMA transfer specified in the ML555 GUI.
- Writes a `1` to DMACST[2] to initiate the DMA read operation.
- Polls the Read DMA Done bit to determine when the DMA operation is complete.
- Computes the DMA read performance based upon the user-requested transfer size and read-only counter values in the register file. Outputs the result to the GUI log window.

Full duplex DMA transfers can be executed by writing a `1` to DMACST[0] and DMACST[2] simultaneously. To keep both transmit and receive channels full, the hardware monitors the state of the channels. For full duplex transfers of 4 KB and under, typically, the hardware executes eight non-posted reads followed by eight posted writes. Full duplex transfers above 4 KB are not executed on any fixed pattern.

Using the host PC processor to poll the register file for status, indicating a DMA read or write completion, can impact the translation layer throughput of this design. After starting a DMA operation, the host PC processor waits for a programmed delay time before polling the endpoint register file to determine if the DMA transfer has completed. The programmed delay time is more than enough time to complete the largest DMA transfer size supported by the design. Polling after the DMA transfer has completed has no impact on the transaction layer throughput of the design.

Several state machines are located in the DMA Control/Status Register File that work in conjunction to feed the Internal Control Block with smaller transfers. These state machines are capable of breaking up a large transfer on the following boundaries: 4 KB, 2 KB, 1 KB, 512 bytes, 256 bytes, and 128 bytes. The state machines break a large transfer request on the largest possible boundaries and transfer a series of smaller requests to the Internal Control Block. For example, a 9600 byte transfer is broken into these smaller transfer sizes:

- 2 x 4 KB
- 1 x 1 KB
- 1 x 256 bytes
- 1 x 128 bytes

The state machines automatically increment the source and destination addresses and transfer size DMA parameters for the smaller sub-transfers and pass these parameters to the Internal Control Block.

Other Support Blocks

Egress Data Presenter

The egress data presenter block provides 64-bit data to the TX engine. Data can be from multiple sources, but it is typically from the DDR2 memory. This block reads 128-bit DDR2 data

out of the DMA-to-DDR2 interface block egress FIFO, pipelines the data, and converts the width to 64 bits. The egress data presenter utilizes a simple handshaking protocol to communicate with the TX engine.

Read Request Wrapper

The read request wrapper serves as a communication port between the TX engine and RX engine. The TX engine passes information about pending non-posted read requests to the RX engine. This information is stored in a dual-port RAM, 32 bits wide by 32 bits deep, and is constructed from distributed RAM using the CORE Generator software. The bits of each entry contain this information:

- [21:0] = DDR2 destination address.
- [31:22] = Length of read request in DWORDS.

The dual-port RAM is addressed using the Tag field from the outgoing non-posted read requests and incoming completion packets. The TX engine writes the initial information to the dual-port RAM for each outgoing non-posted read request. The RX engine uses this information to place the incoming data in the correct location in the DDR2 memory. Further, the RX engine updates the DDR2 Destination Address field in the dual-port RAM after each completion packet that it receives.

In addition to the dual-port RAM, the read request wrapper also contains a small 32 x 1 Shadow RAM. Each entry corresponds to one of the 32 available entries in the dual-port RAM and qualifies the corresponding entry. A value of one means the entry in the dual-port RAM is valid and pending. The TX Engine sets the bit when writing to the dual-port RAM. The RX engine clears the bit upon reception of the last completion packet for a given non-posted read request.

The read request wrapper also contains completion timeout circuitry. This circuitry monitors the Shadow RAM for each new read request and sets a corresponding timer of approximately 25 ms. If the timer expires before the read request is fulfilled, a completion timeout error message is signaled to the root complex.

DDR2 SDRAM SODIMM

The ML555 board contains a single-rank, 72-bit, 256 MB DDR2 SDRAM SODIMM memory. The reference design utilizes a 64-bit datapath. The MIG tool, supplied with the CORE Generator software, generates the DDR2 memory controller used in the reference design. The memory controller and data capture technique are described in *High Performance DDR2 SDRAM Interface in Virtex-5 Devices* [Ref 5]. *Memory Interfaces Made Easy with Xilinx FPGAs and the Memory Interface Generator* [Ref 6] contains a discussion of the MIG design flow. The *MIG User Guide* [Ref 7] contains instructions for getting started with the MIG tool and specific information about Virtex-5 FPGA DDR2 memory interfaces. The MIG tool, version 1.72, does not directly support DDR2 SODIMM interfaces. However, a SODIMM memory controller can be created by generating a DDR2 DIMM controller and then editing the generated MIG tool output. User input parameters in MIG design modules are defined as parameters for Verilog and generics for VHDL and are passed down the design hierarchy. The top-level controller file `mem_interface_top` contains all user input parameters for the controller. Table 5 contains the memory parameters required for the ML555 256 MB DDR2 SODIMM with a 200 MHz clock provided from the programmable clock synthesizer integrated circuit on the ML555 board.

Table 5: **ML555 256 MB DDR2 SODIMM MIG Tool Memory Controller Top-Level Design Parameters**

Memory Controller Parameter	Value	Description
BANK_WIDTH	2	Number of memory bank address bits.
CKE_WIDTH	1	Number of memory clock enable outputs.
CLK_WIDTH	2	Number of memory clock outputs.
COL_WIDTH	10	Number of memory column address bits for 256 MB SODIMM.
CS_NUM	1	Number of separate memory chip selects.
CS_WIDTH	1	Number of total memory chip selects.
CS_BITS	0	Set to $\log_2$(CS_NUM); rounded up to an integer value.
DM_WIDTH	8	Number of data mask bits.
DQ_WIDTH	64	Number of data bits.
DQ_PER_DQS	8	Number of DQ data bits per DQS strobe.
DQS_WIDTH	8	Number of DQS strobes.
DQ_BITS	7	Set to $\log_2$(DQS_WIDTH x DQ_PER_DQS).
DQS_BITS	4	Set to $\log_2$(DQS_WIDTH).
ODT_WIDTH	1	Number of memory on-die termination (ODT) enables.
ROW_WIDTH	13	Number of memory row address bits for 256 MB SODIMM.
ADDITIVE_LAT	0	Additive write latency.
BURST_LEN	4	Burst length in double words.
BURST_TYPE	0	Burst type (0 = sequential, 1 = interleaved).
CAS_LAT	4	CAS latency.
ECC_ENABLE	0	Enable ECC (0 = disable, 1 = enable).
MULTI_BANK_EN	1	Keeps multiple banks open (0 = disable, 1 = enable).
ODT_TYPE	0	ODT (0 = none, 1 = 75Ω, 2 = 150Ω, 3 = 50Ω).
REDUCE_DRV	0	Reduce drive strength for memory I/O (0 = no, 1 = yes).
REG_ENABLE	0	Register address and control (0 = no, 1 = yes).
TREFI_NS	7800	Maximum DDR2 automatic refresh interval in nanoseconds.
TRAS	40000	Minimum time between DDR2 ACTIVE command to PRECHARGE delay (ps).
TRCD	15000	DDR2 ACTIVE command to READ/WRITE delay (ps).
TRFC	127500	Minimum DDR2 REFRESH command to next ACTIVE command delay (ps).
TRP	15000	DDR2 PRECHARGE command period (ps).
TRTP	7500	DDR2 READ command to PRECHARGE delay (ps).
TWR	15000	DDR2 WRITE command to PRECHARGE write recovery time (ps).
TWTR	10000	DDR2 WRITE command to READ command delay (ps).
IDEL_HIGH_PERF	True	Number of taps for DQ IDELAY.

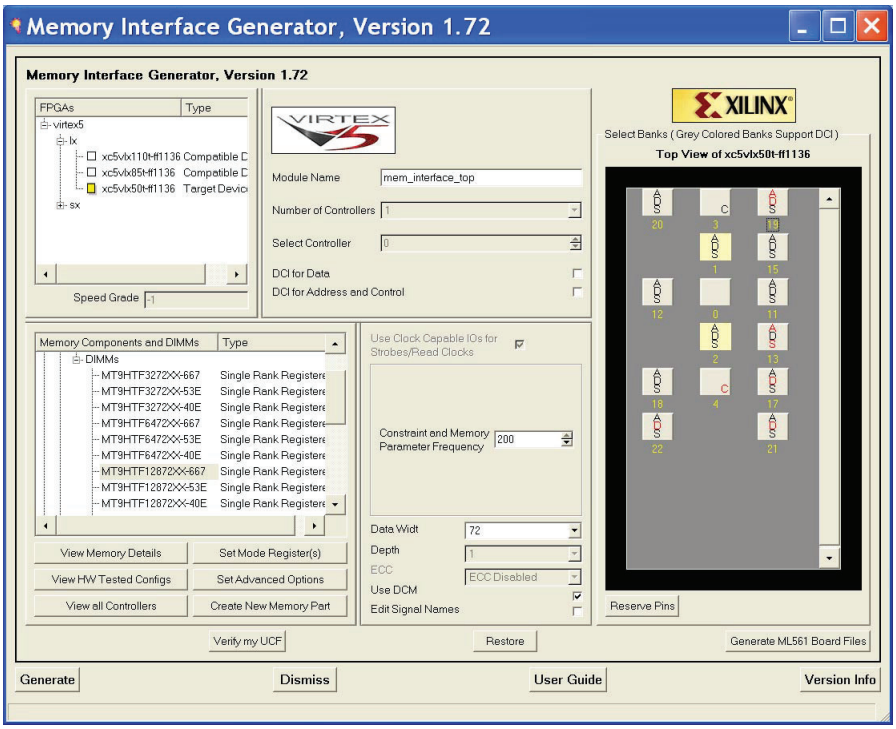
Table 5: **ML555 256 MB DDR2 SODIMM MIG Tool Memory Controller Top-Level Design Parameters *(Cont'd)***

Memory Controller Parameter	Value	Description
SIM_ONLY	0	Set to 1 to skip SDRAM power up delay only when performing logic simulations.
CLK_PERIOD	5000	Core and memory clock period (ps).
RST_ACT_LOW	1	1 = Active-Low reset, 0 = Active-High reset.
DLL_FREQ_MODE	High	Digital Clock Manager (DCM) frequency range. See the *Virtex-5 Data Sheet: DC and Switching Characteristics* [Ref 8] for DCM frequency range specifications by speed grade.

For the latest version of the Virtex-5 FPGA DDR2 controller design, download the IP update from http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp. Figure 16 shows the MIG tool GUI screen used to begin the DDR2 controller design. The XC5VLX50T-FF1136 device is selected during creation of the CORE Generator software project. The main items to verify during regeneration of the DDR2 DIMM design are:

- The top-level module name must use the default `mem_interface_top`.
- Check the **Use Clock Capable IOs** for strobes/read clocks.
- Do not check the **DCI for Data** box.
- Do not check the **DCI for Address and Control** box.
- Select data width of 72 (versus 144). This is later modified in the manual parameter edit task.
- Select constraint and memory parameter frequency of 200 MHz.
- Select **Use DCM**.
- Select any banks for address, data, and control. The MIG tool does not allow the selection of banks that are used on the ML555 board. The tool outputs an error if banks are not selected. The reference design contains location constraints (LOCs), which specify the locations of the pin assignments and other design criteria for the ML555-specific controller.
- Select **Generate**.

The generated output files are placed in `<project_directory>\mem_interface_top_withouttb\rtl`.



X859_16_110907

Figure 16: **MIG Tool GUI Screen for Virtex-5 FPGA DDR2 DIMM Controller**

Two directories are created, one with a synthesizable testbench and one without. The design files used in this reference design are copied from the RTL directory without a testbench. Figure 17 shows the top-level block diagram of the MIG DDR2 controller component. The *MIG User Guide* [Ref 7] contains a detailed description of the interface signals, component hierarchy, and low-level design modules.

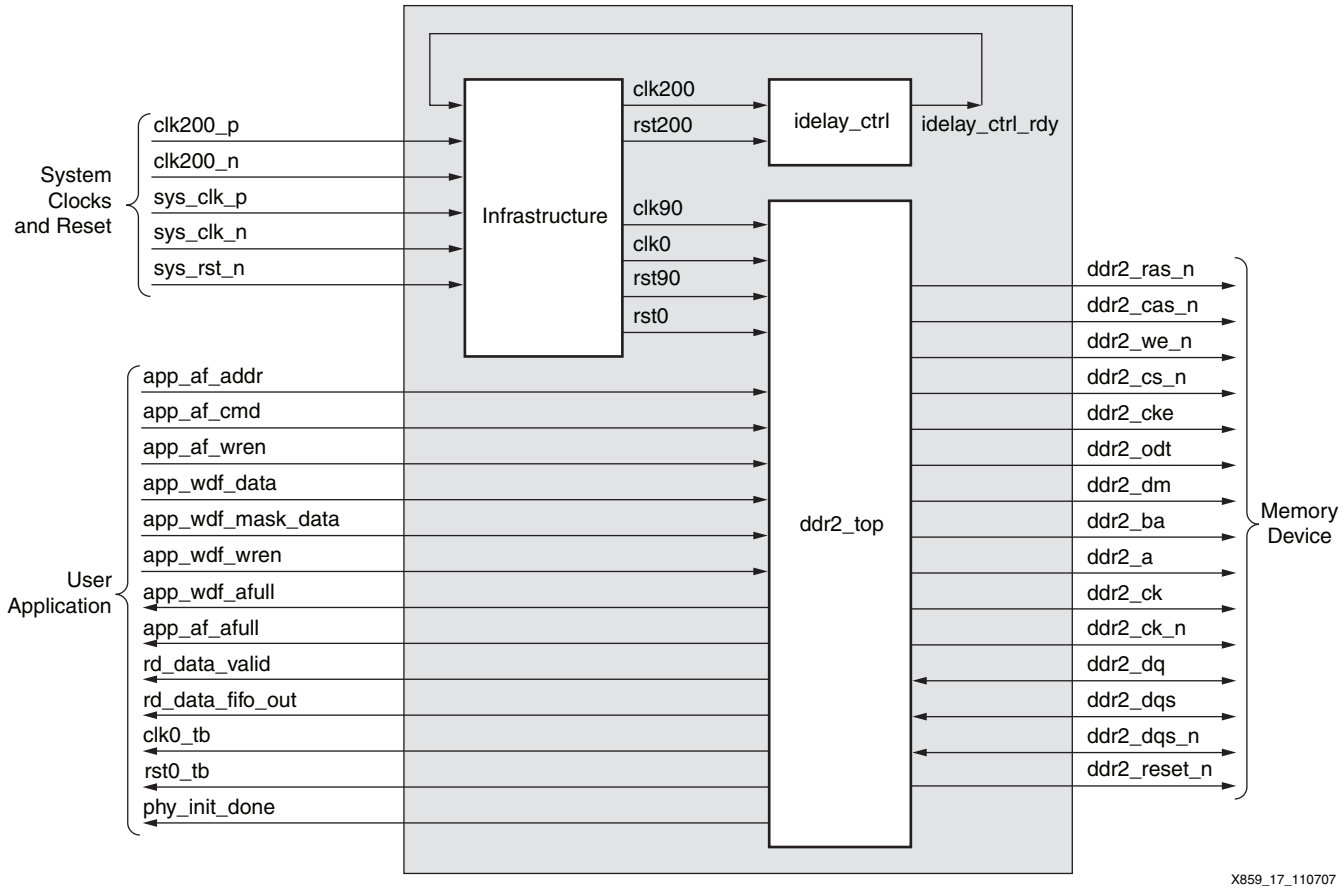


Figure 17: **MIG DDR2 Controller with DCM without a Testbench**

When using the MIG tool to upgrade or modify the DDR2 memory controller for this design, several steps must be completed to generate a controller that can function on the ML555 development board for PCI Express designs:

1. Create a Virtex-5 FPGA DDR2 memory controller using the DDR2 DIMM module MT9HTF12872.
2. Replace the contents of the RTL directory from the reference design ZIP file with the contents of the RTL directory generated by the MIG tool.
3. Edit the top-level memory controller HDL file `mem_interface_top`. Change or verify memory parameters (Verilog design) or generics (VHDL design) to match those listed in Table 5. The controller uses a Column Address Select (CAS) latency of 4, whereas the ML555 board has a SODIMM with CAS latency of 5. When operating the SODIMM at a clock frequency of 200 MHz, a CAS latency of 4 is acceptable.

The ML555 board contains a programmable clock synthesizer integrated circuit that provides the source clock for the DDR2 memory controller. This permits the integrated Endpoint block for PCI Express designs to run at a different frequency than the DDR2 controller. For example, a 4-lane Endpoint design uses a local clock of 125 MHz and an 8-lane Endpoint design uses 250 MHz. For this design, the ML555 board is set up to have the ICS1 clock synthesizer running at 200 MHz, which provides a DDR2 rate of 400 Mb/s per data pin. Refer to the *Virtex-5 ML555 Development Kit for PCI and PCI Express Designs User Guide* [Ref 9] for additional information on setting up the ML555 board.

The DDR2 memory is 64 bits wide, providing a burst transfer rate of 3200 MB/s, which is more than sufficient for an 8-lane PCI Express design with a maximum half duplex burst transfer rate

of 2000 MB/s. Buffering is provided in the FPGA such that the DDR2 memory does not overrun the capability of the Endpoint block to transmit data.

Clocks

This design uses three input clocks available on the ML555 board:

- A 200 MHz Low-Voltage Positive Emitter Coupled Logic (LVPECL) clock is used as the reference clock for the IDELAYCTRL element in the FPGA.
- A programmable clock synthesizer is used as the source for the memory controller. This clock is programmed using selectable switch inputs on the ML555 board. The frequency synthesizer can synthesize frequencies between 31.5 MHz and 700 MHz, in increments of 1.5 MHz. This design assumes a clock input of 200 MHz for the DDR2 controller.
- The 100 MHz PCI Express reference clock is input to the GTP_DUAL tile location X0Y2 of the ML555 board. This clock is used as the reference clock for the transceiver and is synthesized to provide the user clock to the integrated Endpoint block and user application logic. For an eight-lane Endpoint design, the synthesized clock is 250 MHz.

Ports

The top-level design input and output ports are listed in Table 6. Refer to the UCF file included with the reference design for detailed design constraint information, including FPGA pin assignments for the ML555 development board.

Table 6: **Top-Level Design Input and Output Ports**

Signal Name	I/O	Width	Description	Interfaces With
REFCLK_P REFCLK_N	I	Differential clock pair	AC-coupled, 100 MHz reference clock input from a PC system unit with PCI Express add-in card connector to ML555 GTP transceiver instance X0Y2.	PCI Express root complex.
RXP RXN	I	8 differential pairs	AC-coupled PCI Express root complex transmitter outputs to Endpoint receiver inputs from add-in card connector.	
TXP TXN	O	8 differential pairs	AC-coupled Endpoint block for PCI Express transceiver outputs to root complex receiver inputs to add-in card connector.	
sys_reset_n	I	1	Active-Low system reset input source from the system unit add-in card connector pin PERST#. This signal indicates that the system power supply is within acceptable tolerances and the voltage is stable. Master reset for FPGA design elements.	System board reset signal PERST#.

Table 6: **Top-Level Design Input and Output Ports *(Cont'd)***

Signal Name	I/O	Width	Description	Interfaces With
CLK200_P CLK200_N	I	Differential clock pair	ML555 200 MHz LVPECL clock input for IDELAYCNTRL reference clock.	ML555 DDR2 SODIMM and DDR2 memory controller.
SYS_CLK_P SYS_CLK_N	I	Differential clock pair	ML555 programmable LVDS clock synthesizer input used for DDR2 memory controller clock.	
PLOAD_1	O	1	When asserted, performs a parallel load of the ICS1 clock synthesizer on the ML555 board. This eliminates the need to push and release SW9 on the ML555 board to generate the 200 MHz SYS_CLK for the DDR2 controller.	
DDR2_CK DDR2_CK_N	O	Differential clock pair	DDR2 clocks to SODIMM from FPGA.	
DDR2_DQ	I/O	64	DDR2 memory data. Only 64 of 72 bits are used on the SODIMM interface of the ML555 board. Data bits are paired with appropriate DQS strobes in a bank.	
DDR2_DQS	I/O	8	DDR2 memory data strobes, one per byte. Must use clock-capable I/O pins on the FPGA.	
DDR2_DM	O	8	DDR2 memory data mask, one per byte.	
DDR2_A	O	13	DDR2 multiplexed row and column address bits.	
DDR2_BA	O	2	DDR2 bank address.	
DDR2_RAS_N	O	1	DDR2 row address select.	
DDR2_CAS_N	O	1	DDR2 column address select.	
DDR2_WE_N	O	1	DDR2 write enable.	
DDR2_CS_N	O	1	DDR2 chip select.	
DDR2_RESET_N	O	1	Active-Low DDR2 reset.	
phy_initialization_done_n	O	1	Asserted when the DDR2 memory controller physical layer initialization sequence is finished. The controller user interface is ready to receive read and write commands to access memory. This signal is routed to an ML555 board user LED for a visual indicator of memory physical interface status.	ML555 user LED indicator (USER_LED1).
LED_link_up_n	O	1	Asserted low when the PCI Express link is active.	ML555 user LED indicator (USER_LED0).

Design Limitations

This design is intended to be used by system integrators to test the performance of the PCI Express link in a system environment. Facilities are provided to test the read or write DMA performance. The maximum read and write DMA transfer size supported by the software application is given by Equation 1:

$$\text{Number of bytes to transfer} = 128 \times 2^{n} \qquad \textit{Equation 1}$$

where n = 0 to 13

The Endpoint Block Plus Wrapper for PCI Express designs supports a maximum payload size of 512 bytes. In an open system, this is not a concern because the configuration space device control register for PCI Express designs typically limits the maximum payload size to 128 bytes. The reference design does not support ECRC in the TLP packet.

ML555 GUI and WinDriver-Based Software Application

In protected operating systems such as Windows and Linux, a software developer cannot access hardware directly from the application level or user mode, where development work is normally done. Hardware can only be accessed from within the operating system itself, utilizing software modules called device drivers. Jungo, Ltd., offers the WinDriver driver development toolkit that enables software developers to quickly create custom device drivers that can run on a multitude of operating systems without modification. The ML555 DMA control and status device driver and application software utilize the WinDriver development toolkit from Jungo. The custom driver associated with this reference design and ML555 board provides these functions:

- Scans the computer system for attached peripheral devices
- Allocates a system memory buffer for ML555 DMA operations from the Endpoint
- Initializes system memory buffer with specific data contents
- Displays system memory buffer upon request

To learn more about the Jungo WinDriver driver development toolkit, visit www.jungo.com. To learn about the technical capabilities of the WinDriver toolkit, go to the Jungo technical support site at www.jungo.com/support/support_windriver.html. The reference design includes a device driver developed using a licensed version of the WinDriver development kit. A Jungo WinDriver license is not required to run this reference design in a Windows XP Professional system environment.

Design Implementation

The Virtex-5 FPGA reference design is implemented using ISE software, version 10.1.02.

LogiCORE IP

Table 7 shows the LogiCORE IP and versions used in the reference design.

Table 7: **LogiCORE IP Used in the Reference Design**

LogiCORE IP	Version	Design Usage
Virtex-5 FPGA Endpoint Block Plus wrapper for PCI Express	1.8	Endpoint interface for PCI Express designs.
DDR2 Memory Controller	MIG tool (v1.72)	Single-rank, DDR2, SODIMM physical interface and controller.
FIFO	3.3 and 3.4	User application logic FIFO buffers.

Notes:

1. Virtex-5 FPGA Endpoint Block Plus wrapper is free but requires a click-through license from Xilinx.
2. The DDR2 memory controller is free but requires a click-through license from Xilinx.

Simulation

The reference design is simulated using ModelSim SE 6.3c and uses a modified version of the downstream port testbench model from the integrated Endpoint block for PCI Express. For more information on the integrated Endpoint block's downstream port testbench model, see Chapter 5 of the *Virtex-5 Integrated Endpoint Block for PCI Express Designs User Guide* [Ref 10].

The prerequisites for the simulation environment used for this reference design are:

- ModelSim SE 6.3c or equivalent
- The Xilinx UniSim, SimPrim, SecureIP, and XilinxCoreLib for Verilog are compiled into a simulation library

To run the simulation, modify `modelsim.ini`, located in the `/xapp859/fpga/simulation/functional` directory to point to the correct library location for UniSim, SimPrim, SecureIP, and XilinxCoreLib. For example:

```
UNISIMS_VER = C:/Xilinx/verilog/mti_se/unisims_ver
SIMPRIMS_VER = C:/Xilinx/verilog/mti_se/simprims_ver
XILINXCORELIB_VER = C:/Xilinx/verilog/mti_se/XilinxCoreLib_ver
SECUREIP = C:/Xilinx/verilog/mti_se/secureip
```

After the simulation environment is set up correctly, run the `tb.fdo.bat` script by double-clicking it. This launches ModelSim and runs the `tb.fdo` script to launch the simulation. The `tb.fdo.bat` file is located in the `/xapp859/fpga/simulation/functional` directory.

Implementation Results

Table 8 shows the resource utilization in the reference design as implemented on the ML555 board. See *Endpoint Block Plus for PCI Express Product Specification* [Ref 1] for a design resource utilization summary of the Endpoint Block Plus Wrapper LogiCORE product.

Table 8: **Design Resource Utilization**

Resource	Used	Available	Utilization (%)
Slice Registers	10,523	28,800	36
Slice LUTs	8,716	28,800	30
DCM	1	12	8
PLL_ADV	1	6	16
Block RAM	17	60	28
IOB	123	480	25
GTP_DUAL	4	6	66
PCIe	1	1	100

Design Files

The design is organized as shown in Figure 18. The MIG DDR2 controller details have not been expanded and can be found in the *MIG User Guide* [Ref 7]. The `readme.txt` file included with the reference design ZIP file includes additional information on project files used for implementation and simulation purposes.

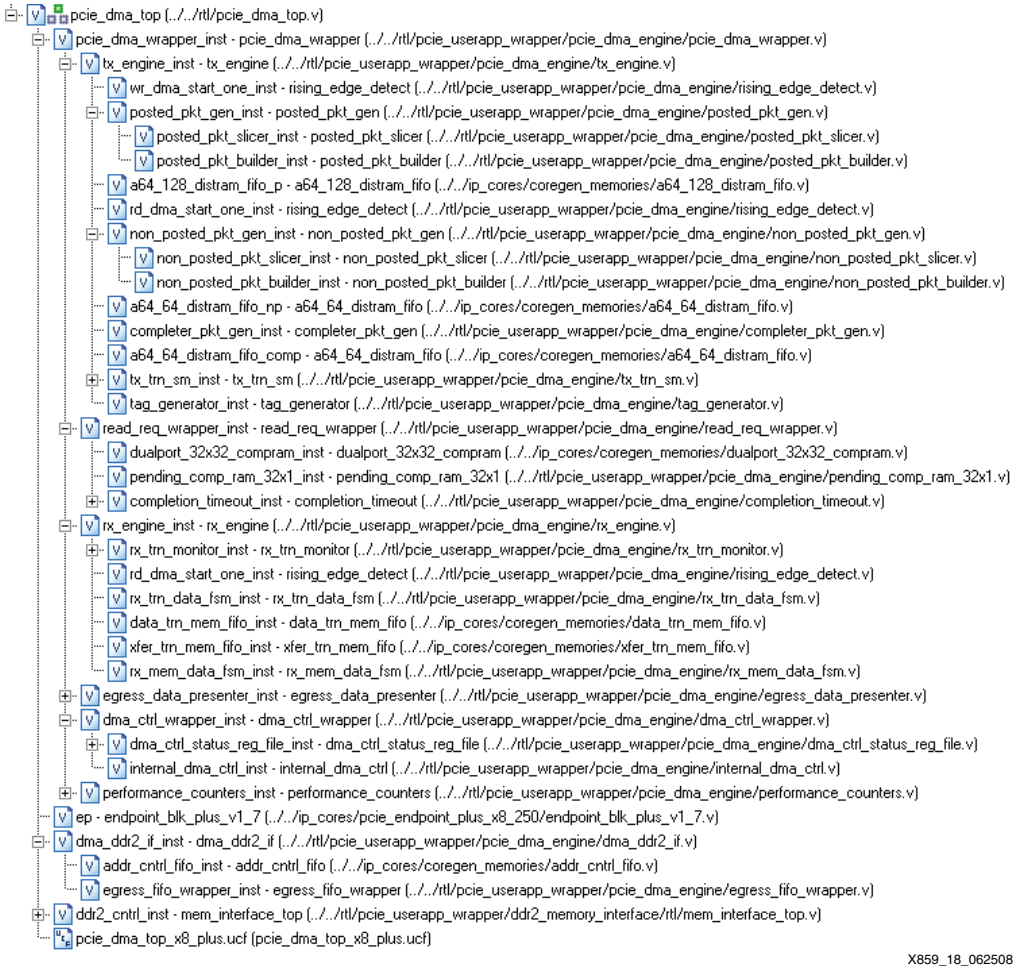


Figure 18: **Design Hierarchy**

Running the Demonstration Design in a PC System Unit

Running the Reference Design Demonstration: Quick Start Guide

This section summarizes the steps required to install and run the design in the ASUS P5B-VM motherboard. Detailed step-by-step instructions are included in the sections that follows this summary. The ASUS P5B-VM has a 16-lane PCI Express add-in card connector that the ML555 board plugs into.

1. Set up/verify the ML555 board settings:
   a. Power supply settings to PCIe mode
   b. Programmable clock synthesizer dip switch SW10 and SW12 to 200 MHz
   c. Configuration Mode switch SW5 to Master SelectMAP
2. Plug the ML555 board into the motherboard.
3. Connect a JTAG programming cable to P5 on the ML555 board, apply power to the host motherboard system (named the host system), and program the U1 platform flash with `xapp859.mcs`, using iMPACT or a similar JTAG program. U1 is the third device in the JTAG chain. After programming, cycle the power to the host system.
4. The reference design application software utilizes Microsoft .NET Framework version 2.0. Verify that the correct version is installed on the PC or install a free upgrade. (See "Installing Microsoft .NET Framework Version 2.0," page 36).
5. Install the Jungo device driver `WD910_csharp_xapp859.sys` on the host system. (See "Installing the ML555 Jungo Device Driver on Windows," page 39.) Remove power from the host system after installation.
6. Apply power to the host system.
7. Start the ML555 GUI application `WD910_csharp_xapp859.exe` on the host system. (See "Running the DMA Demonstration User Application," page 41.)
8. Setup is concluded and the demonstration is ready to use.

The reference design trains down to the lane width that the host system supports on the add-in-card connector slot that the ML555 is plugged into.

PC Requirements

These elements are required to run the reference design:

- A PC with Microsoft Windows XP SP2
- One available 8-lane (or 16-lane) PCI Express Endpoint, full-size, add-in, card slot for the ML555 board

The reference design is system verified on an ASUS P5B-VM motherboard with an Intel 965 chipset, Windows XP Professional, and an ML555 board containing an XC5VLX50T-1FFG1136C device. The ML555 board is installed in a 16-lane PCIe add-in card connector in the ASUS motherboard. The motherboard contains an integrated VGA port for monitor support, freeing up the 16-lane PCIe interface for the ML555 Endpoint design.

Before installing the ML555 board in the PC system unit, the ML555 Board Setup section of the *Virtex-5 ML555 Development Kit User Guide* should be reviewed [Ref 9]. The ML555 board must be verified to be configured for PCI Express compliant system power before applying power to the PC and the ML555 board.

ML555 Board Setup

The *Virtex-5 ML555 Development Kit User Guide* [Ref 9] should be consulted before installing the ML555 board into the system unit. These items should be verified before installing the ML555 board in the PC system and applying power to the system unit and ML555 board:

- The ML555 board is configured to run off the PCI Express compliant power supply (versus the PCI or PCI-X bus power) as shown in Figure 19.

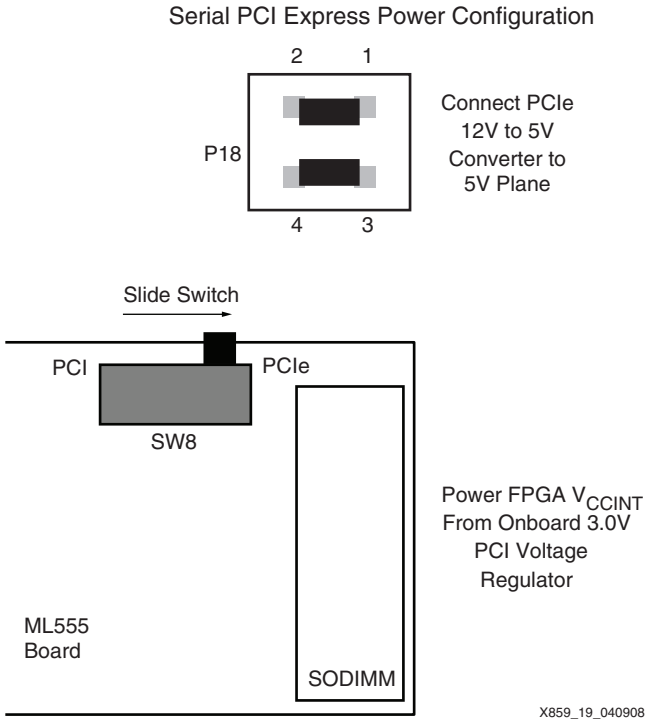


Figure 19: **ML555 Power Configuration**

- The ML555 board programmable clock synthesizer SW10 and SW12 switch inputs are set to generate a 200 MHz clock for the DDR2 SODIMM according to Figure 20. Figure 21 shows the location of SW10 and SW12 on the ML555 board. It is not necessary to push and release the parallel load SW9 because the reference design automatically asserts the parallel load input of the clock synthesizer module.

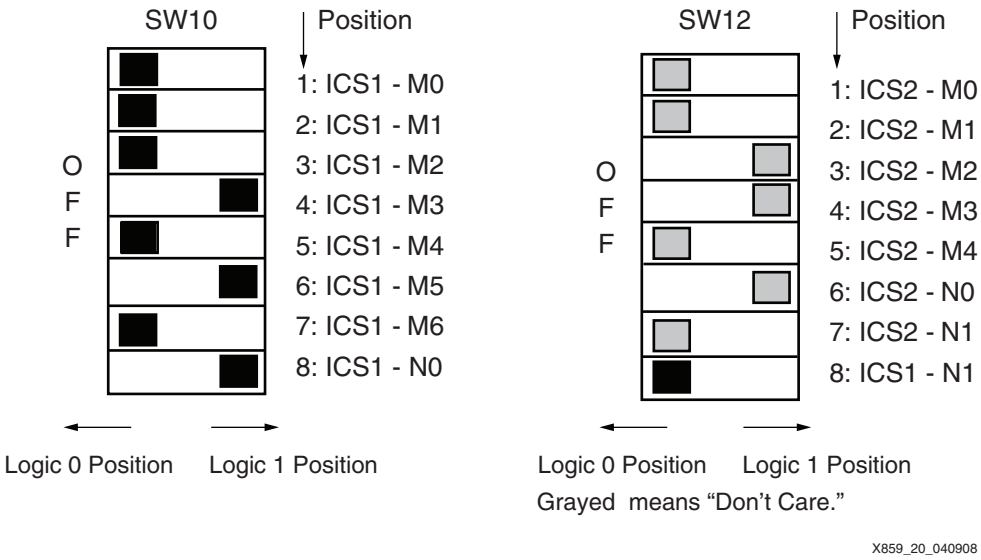


Figure 20: **SW10 and SW12 Switch Inputs**

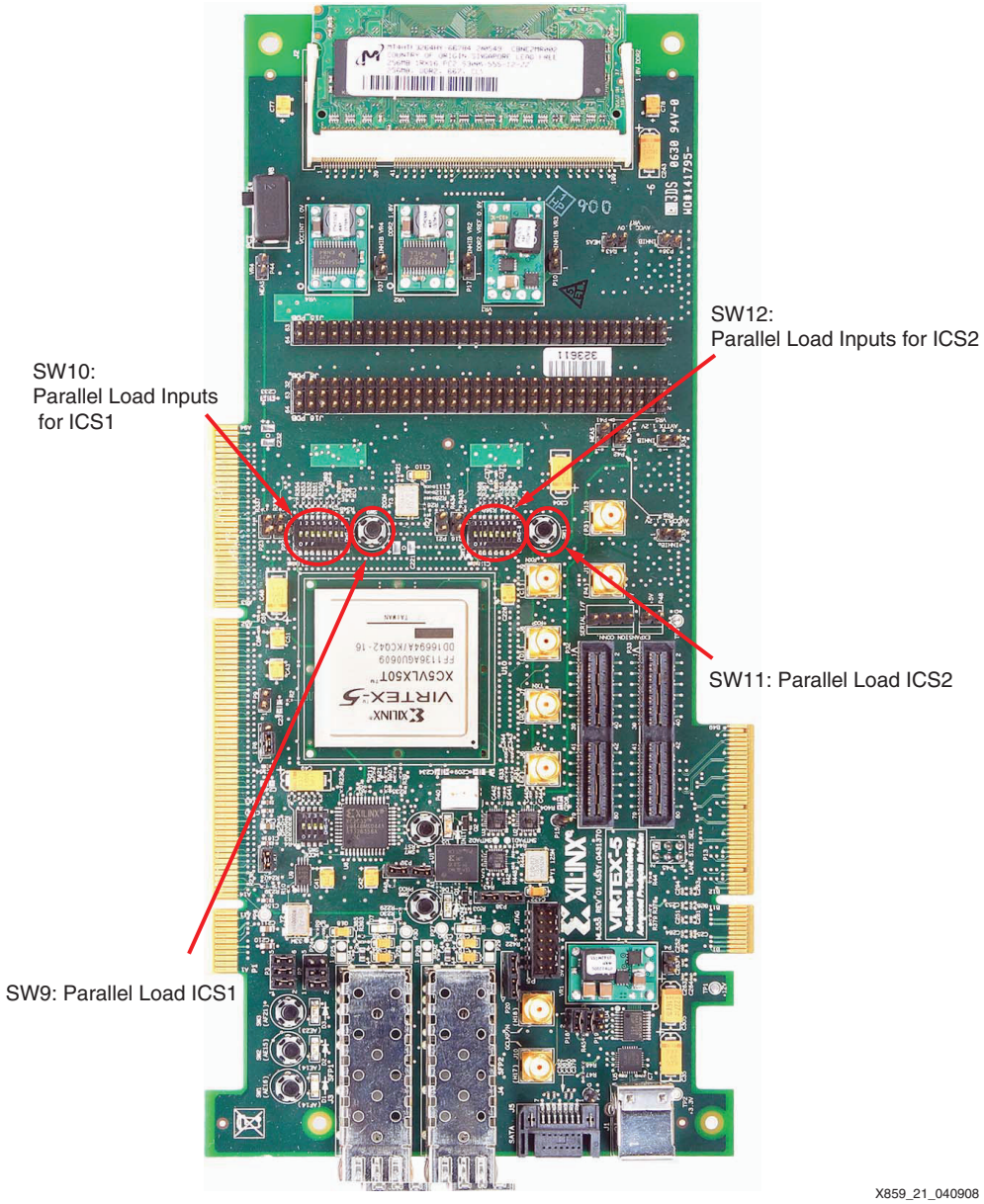


Figure 21: **ML555 Clock Synthesizer 1 Parallel Load Inputs SW10 and SW12 Locations**

- The configuration Mode switch SW5 is set to configure the FPGA from platform flash according to default settings of the ML555 board. Set to Master SelectMAP mode (Figure 22).

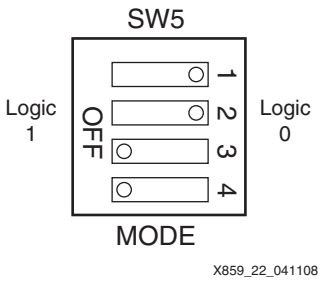


Figure 22: **Configuration Mode Switch**

- Platform flash U1, revision 0 is selected to configure the FPGA. All three shunts need to be installed on header P3. Platform flash U1 is programmed with the reference design as part of the setup procedure.

Proper operation of this reference design requires headers on the ML555 board to have shunts installed in accordance with Table 9. These shunts are provided with the ML555 board. In most cases, Table 9 reflects the default configuration of the ML555 board, as built. Optional headers for PCI and PCI-X applications are listed for completeness. Prior to installing the ML555 into the PC, the proper configuration of the board headers must be verified. Any header on the ML555 not listed in Table 9 should be left as an open circuit.

Table 9: **ML555 Shunt Installation Requirements for XAPP859 Operation**

ML555 Board Reference Designator	Connectivity for XAPP859 PCI Express Application	Comments	Function Provided
P2	P2-1 to P2-2 (No other connections)	Required	FPGA drives CCLK to both platform flash devices for Master SelectMAP configuration at system power up.
P3	P3-1 to P3-2 P3-3 to P3-4 P3-5 to P3-6	Required	Platform flash image selection defaults to select U1 image 0 for default FPGA configuration via SelectMAP. See *Virtex-5 FPGA ML555 Development Kit for PCI and PCI Express Designs User Guide* [Ref 9] for alternative platform flash selection options.
P7	P7-1 Install shunt but do not short to P7-2	Optional(1)	Only used in PCI and PCI-X modes. Provides capability for PCI and PCI-X users to control PME_B connection.
P8	P8-2 to P8-3	Optional(1)	PCI Mode (not PCIX).
P9	P9-1 to P9-2	Optional(1)	PCI Mode at 0–33 MHz.
P18	P18-1 to P18-2 P18-3 to P18-4	Required	Configure DC power for PCI Express mode. See Figure 19.

Table 9: **ML555 Shunt Installation Requirements for XAPP859 Operation *(Cont'd)***

ML555 Board Reference Designator	Connectivity for XAPP859 PCI Express Application	Comments	Function Provided
P19	Open circuit	No connection	P19 is used to determine power consumption of the 12V to 5V power regulator on the ML555. See *Virtex-5 FPGA ML555 Development Kit for PCI and PCI Express Designs User Guide* [Ref 9] for further information.
P20	P20-1 to P20-2 P20-3 to P20-4	Required	Enable platform flash device U15 in the JTAG configuration chain.
P38	P38-1 to P38-2 P38-3 to P38-4	Required	Enable platform flash U1 in the JTAG configuration chain.
P39	P39-1 to P39-2 P39-3 to P39-4	Required	Enable CPLD U6 in the JTAG configuration chain.
P41, P42, P43, P44	Open circuit	No connections	P41, P42, P43, and P44 are used to determine power consumption of the AVTTX, AVCCPLL, AVCC, and V_{CCINT} FPGA power supplies on the ML555 board. See *Virtex-5 FPGA ML555 Development Kit for PCI and PCI Express Designs User Guide* [Ref 9] for further information.
P45	P45-1 to P45-2	Suggested(2)	Eight lane PCI Express add-in card function (only required if PC system supports add-in-card sensing.)

Notes:

1. Optional shunt installation. Does not affect operation of this reference design.
2. Suggested shunt installation. Not all PC systems perform sensing operations to determine width of PCI Express add-in-cards. The reference design may work with other settings of P45. See Virtex-5 FPGA ML555 Development Kit for PCI and PCI Express Designs User Guide [Ref 9] for further information.

The reference design ZIP file includes the FPGA and platform flash configuration files `xapp859.bit` and `xapp859.mcs`. The BIT and MCS file are manually loaded into the FPGA and platform flash, respectively, using the JTAG interface of the ML555 board. A script is provided with the reference design which executes iMPACT in batch mode to program the platform flash. See the `readme.txt` file provided in the `xapp859.zip` file for more information.

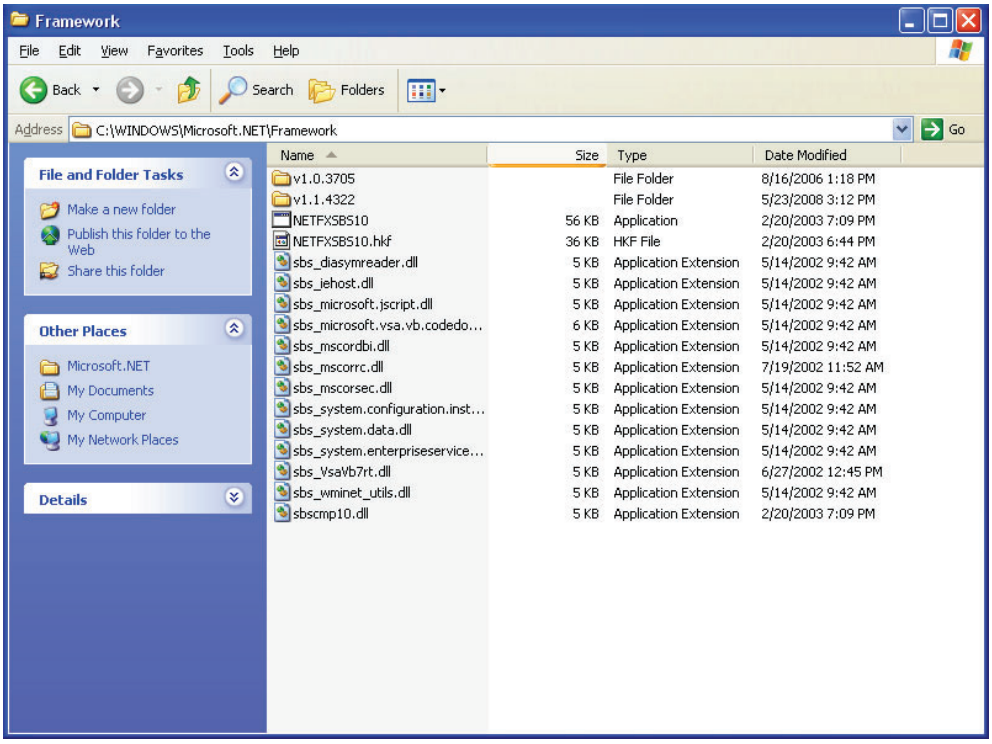
The preferred method to run the reference design is to load the platform flash image of the ML555 board, and cycle the power off and on. After the image is loaded into the platform flash, the reference design is automatically loaded at power on, and the design runs in the system environment without having to restart the computer. The PC system firmware enumerates the ML555 board and configures the add-in card adapter as part of the system initialization process.

The *Virtex-5 ML555 Development Kit User Guide* [Ref 9] contains a detailed board and FPGA configuration section that should be consulted for information about how to load the BIT and MCS files on the ML555 board. When running BITGEN to produce a design image for the ML555 board, a CCLK frequency of 20 MHz must be used, rather than the default 2 MHz. For example:

```
bitgen -g ConfigRate:20 <infile>
```

Installing Microsoft .NET Framework Version 2.0

The reference design application software utilizes Microsoft .NET Framework version 2.0. To determine the version of Microsoft .NET Framework installed on the PC, navigate to the `C:\WINDOWS\Microsoft.NET\Framework` directory. Figure 23 demonstrates a PC with Microsoft Windows XP that does not have Framework version 2.x installed. This PC must be upgraded prior to running the DMA demonstration application. Figure 27 demonstrates a PC with Framework version 2.0 installed that does not require a software upgrade.



X859_23_062608

Figure 23: **PC without Microsoft .NET Framework Version 2.x Installed (Upgrade Required)**

A free software upgrade and instructions for installing the upgrade are available at the Microsoft Download Center Web site:

http://www.microsoft.com/downloads/details.aspx?FamilyID=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=en

Figure 24 shows the initial setup screen for the Framework upgrade installation. Click **Next** to begin Framework 2.0 installation on the PC hosting the ML555 board.

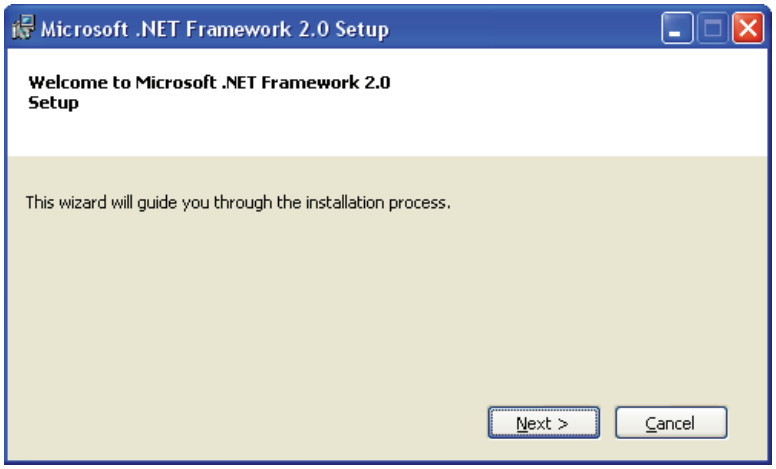


Figure 24: **Microsoft .NET Framework 2.0 Setup Screen**

Figure 25 shows the End-User License Agreement screen. Review and check the box next to "I accept the terms of the License Agreement." Click **Install** to continue with the installation.

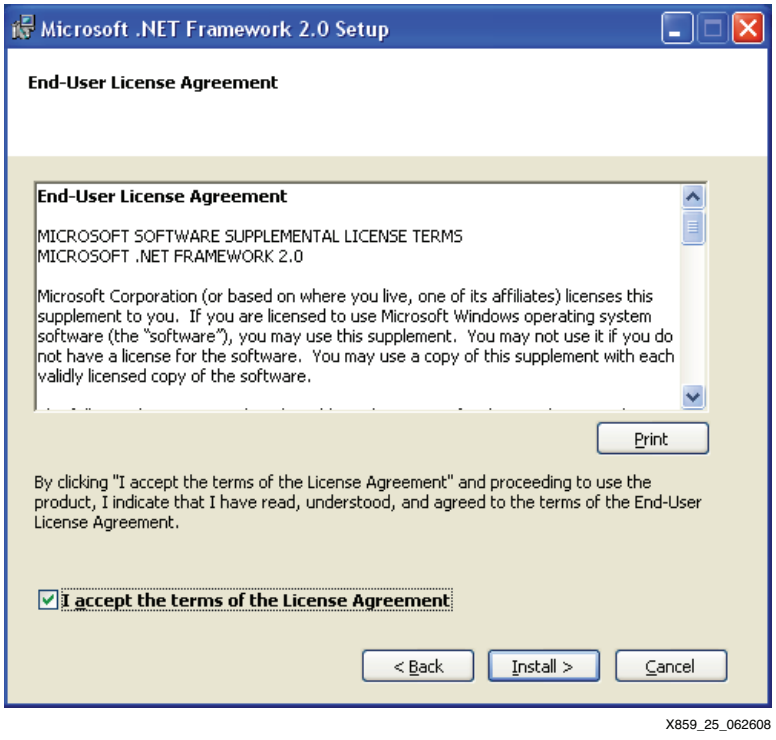


Figure 25: **Review and Accept End-User License Agreement**

When the software upgrade installation is complete, the Setup Complete screen is displayed (Figure 26). Click **Finish** to complete the software upgrade.

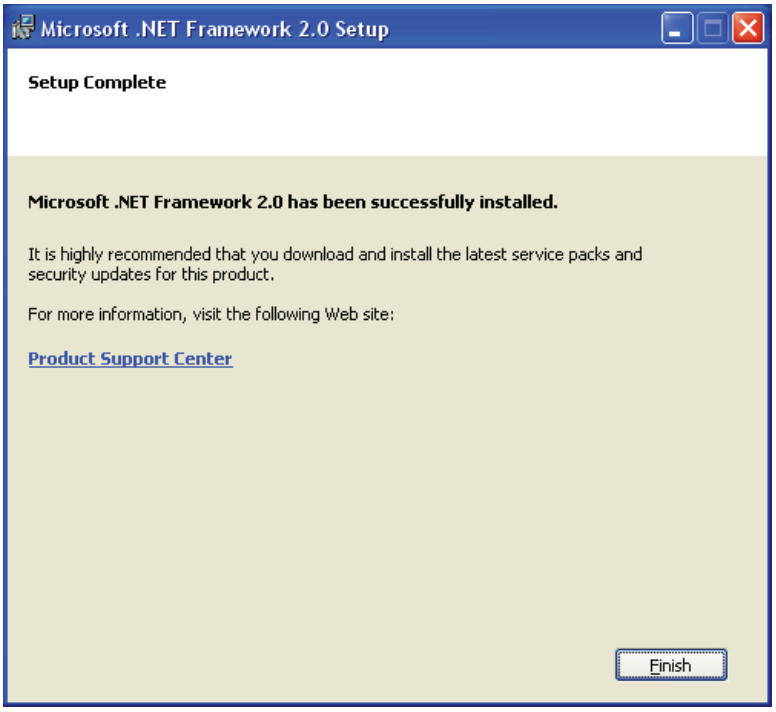


Figure 26: **Microsoft .NET Framework 2.0 Setup Complete**

Figure 27 shows the `C:\WINDOWS\Microsoft.NET\Framework` directory after the version 2.0 software upgrade is complete.

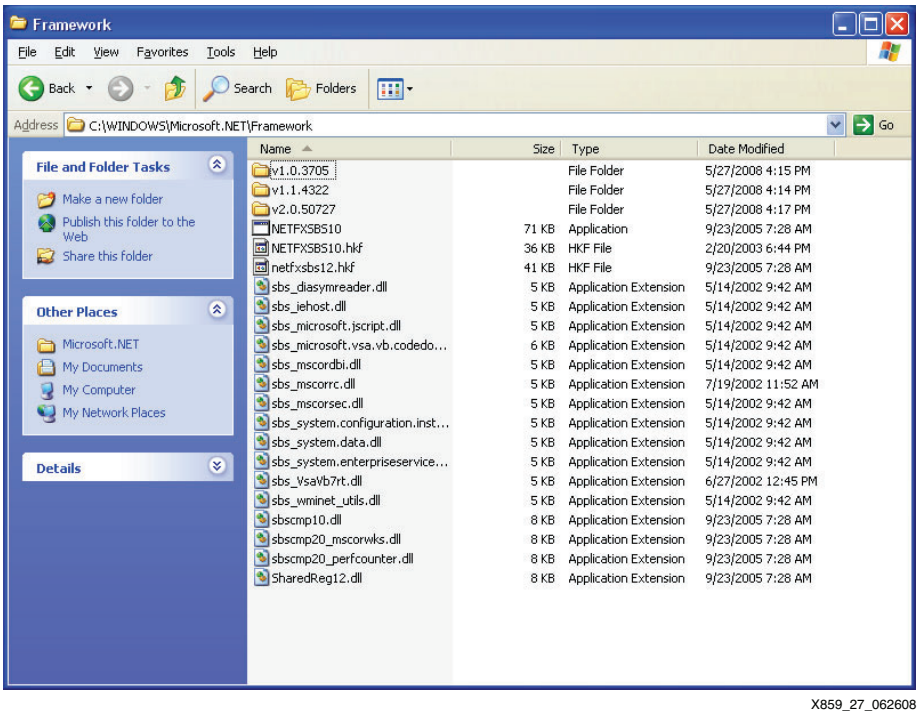


Figure 27: **Microsoft .NET Framework Version 2.0 Installed**

Installing the ML555 Jungo Device Driver on Windows

For the DMA demonstration to work properly, a memory aperture is required on the host system. On Windows-based systems, only a device driver can access protected kernel space functions to acquire a memory buffer and report a physical memory address to a user application. To accomplish this, Xilinx uses the Jungo WinDriver [Ref 11] tools to build a driver and create a user application. This section describes how to install the driver.

The reference design ZIP file contains a directory named `/xapp859/driver`. This directory contains the driver files, installation scripts, user application, and any necessary `.dll` files to run the demonstration application. Table 10 details these files and their use.

Table 10: **Driver Directory File Descriptions**

File Name	Description
`difxapi.dll`	Library used by `wdreg.exe`
`install_WD910_csharp_xapp859_driver.bat`	Installation script for driver
`RichTextBoxPrintCtrl.dll`	Library used by `WD910_csharp_xapp859.exe`
`uninstall_WD910_csharp_xapp859_driver.bat`	Uninstallation script for driver
`WD910_csharp_xapp859.exe`	User application executable
`WD910_csharp_xapp859.sys`	Device driver
`WD910_csharp_xapp859_device.inf`	Device information file
`WD910_csharp_xapp859_driver.inf`	Driver information file
`wdapi910.dll`	Library used by `WD910_csharp_xapp859.exe`
`wdreg.exe`	Driver installation utility

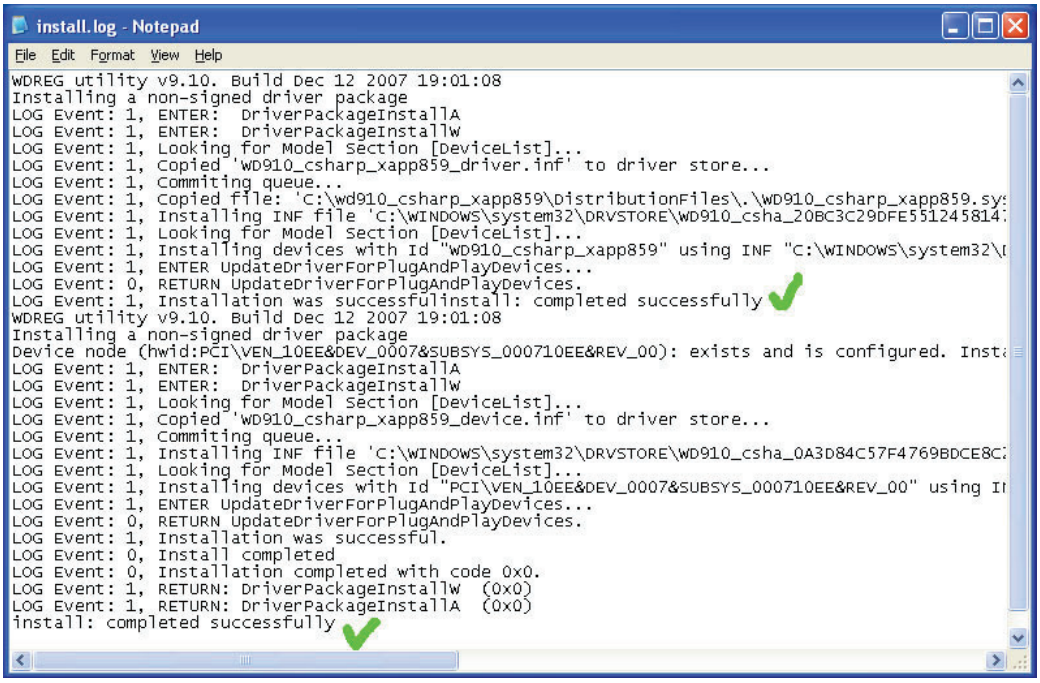
This procedure assumes that the ML555 board has been properly configured for PCI Express compliant system power, the reference design has been programmed into the platform flash configuration device on the ML555 board, and the ML555 board is configured to load the reference design into the FPGA from platform flash at power-up.

1. With the PC powered off, install the ML555 board in an 8-lane or 16-lane add-in-card socket for PCI Express in the PC.
2. Power up the PC and ML555 system before continuing with the DMA demonstration driver installation.
3. Upon power-up, Windows might display the Found New Hardware wizard (Figure 28). Click **Cancel** to exit the wizard.

X859_28_062608

Figure 28: **New Hardware Wizard**

4. Navigate to the `/xapp859/driver` directory and run `install_WD910_csharp_xapp859_driver.bat` by double-clicking the file. This installs the driver. Successful installation can be verified in two ways:
   - Examine the `install.log` file created in the `/xapp859/driver` directory (Figure 29).



XAPP859_29_062608

Figure 29: **`Install.log` File**

- Examine the Windows device manager and verify that the Jungo DEVICE and WD910_csharp_xapp859 entries are listed (Figure 30).

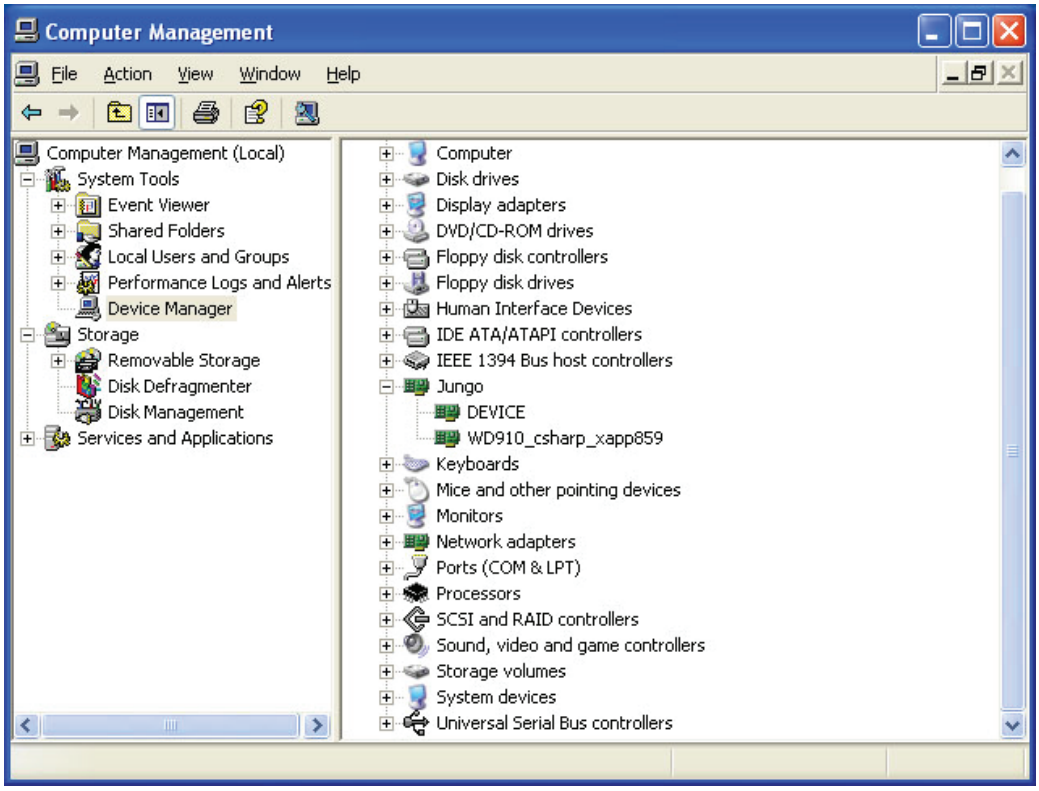


Figure 30: **Device Manager**

5. Reboot the system.

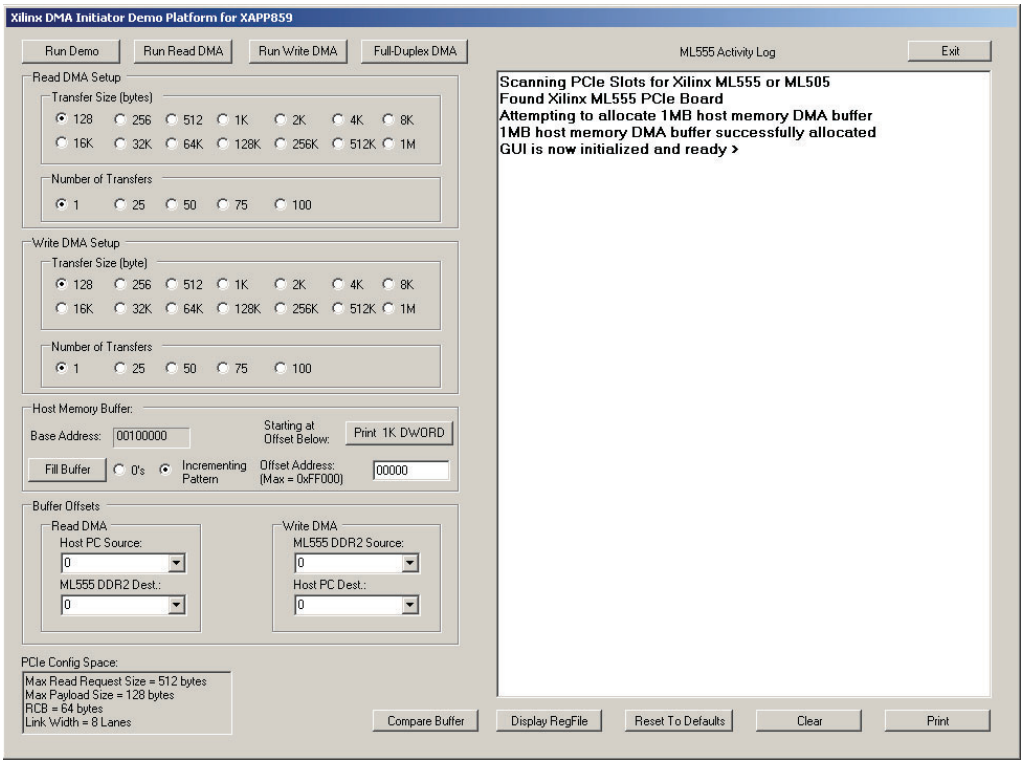
To uninstall the driver, navigate to the `/xapp859/driver` directory and run `uninstall_WD910_csharp_xapp859_driver.bat`. Reboot the system when completed.

Running the DMA Demonstration User Application

To create a desktop application icon, navigate to the `/xapp859/driver directory`, right-click `WD910_csharp_xapp859.exe`, and select **Create Shortcut** from the dropdown menu. Drag the resulting `Shortcut to wd910_csharp_xapp859.exe` file to the desktop and rename it to `ML555 GUI`.

The `WD910_csharp_xapp859.exe` file requires the `wdapi910.dll` library. If `WD910_csharp_xapp859.exe` is moved to a different directory, `wdapi910.dll` must also be moved to that directory. Alternatively, `wdapi910.dll` can be moved to a directory that is located in the `%PATH%` variable.

Upon double-clicking the `ML555 GUI` desktop icon, the application control interface GUI shown in Figure 31 is displayed.



X859_31_062608

Figure 31: **ML555 Demonstration User Application GUI**

Upon launching the ML555 GUI application, several behind-the-scenes activities take place:

- The application cycles through the PCI bus tree in search of an ML555 or ML505 device. This step is reflected by the GUI log window displaying the message `Scanning PCIe Slots for Xilinx ML555 or ML505`.
- After the device is located, it is opened and the driver is associated with the device, as indicated by the GUI log window displaying these messages:
  - `Found Xilinx ML555/ML505 PCIe Board`
  - `Trying to open device`
  - `Device opened successfully`
- A 1 MB contiguous memory buffer is allocated, and the entire buffer is initialized with an incrementing data pattern.
- The GUI log window reports:
  - `Attempting to allocate 1 MB host memory DMA buffer`
  - `1 MB host memory DMA buffer successfully allocated`
  - `GUI is now initialized and ready >`
- Upon 1 MB buffer allocation, the buffer base address is displayed in a GUI text box titled `Base Address`.
- The PCIe configuration space in the ML555 endpoint device is interrogated and a few key items are reported to the PCIe Configuration Space text box in the lower left corner of the GUI. Typically, the report looks like:
  - `Max Read Request Size = 512 bytes`
  - `Max Payload Size = 128 bytes`

- RCB = 64 bytes
- Link Width = 8 lanes

Where RCB is the read completion boundary.

If the application fails to find an ML555 board plugged into a PCIe slot in the host PC, the GUI log window displays the error message `Did not find Xilinx PCIe Board!` Also, if the reference design fails to initialize the DDR2 memory, the GUI detects this within the hardware and reports this condition along with debug hints on fixing the problem.

Successful startup results in the final GUI log window displaying the message `ML555 GUI initialized and Ready >`, at which time the user can proceed with DMA operations using the control features provided within the GUI and described in the following section.

DMA Control and Status GUI

This section describes the controls used to set up and initiate DMA read or write requests utilizing this GUI. The main GUI selections are:

- **Run Demo**: This button causes five transfers of each supported transfer size to occur in a read DMA followed by a write DMA. The transfer size is incremented to the next larger value after each read/write DMA pair of transactions is completed, until all transfer sizes have been executed. The average throughput of each transfer is then reported in the log window.
- **Run Read DMA**: This button causes execution of a read DMA operation using the parameters selected in the Read DMA Setup section of the GUI. The average throughput of the transfer(s) is displayed in the log window.
- **Run Write DMA**: This button causes execution of a write DMA operation using the parameters selected in the Write DMA Setup section of the GUI. The average throughput of the transfer(s) is displayed in the log window.
- **Full-Duplex DMA**: This button causes execution of simultaneous read and write DMA operations using the parameters selected in the Read and Write DMA Setup sections of the GUI. If the Number of Transfers parameter is not the same for both DMA directions, the DMAs are cycled simultaneously (full-duplex mode) until the transfer count of the direction with the smaller transfer count value is decremented to zero. The DMAs in the non-zero transfer count direction are then executed in half-duplex mode. The average throughput of the transfer(s) is displayed in the log window.
- Read DMA Setup: The radio buttons in this section allow the selection of read DMA transfer size in bytes and the number of times to repeat the read DMA operation. The parameters chosen in this section are used to specify the DMA transactions executed when the **Run Read DMA** or **Full-Duplex DMA** buttons are clicked.
- Write DMA Setup: The radio buttons in this section allow the selection of write DMA transfer size in bytes, and the number of times to repeat the write DMA operation. The parameters chosen in this section are used to specify the DMA transactions executed when the **Run Write DMA** or **Full-Duplex DMA** buttons are clicked.
- Host Memory Buffer: This section provides features related to the 1 MB host PC memory buffer that was allocated at GUI launch time:
  - Base Address: This read-only text box displays the base address of the allocated memory buffer.
  - **Fill Buffer**: This button causes the 1 MB memory buffer to be filled with one of two data patterns, depending upon which pattern type radio button is selected, **0's** or **Incrementing Pattern**:
    - The **0's** radio button selects the buffer fill DWORD data pattern to be `0x00000000`.

        - The **Incrementing Pattern** radio button selects the buffer fill DWORD data pattern to be incrementing data corresponding to the incrementing offset memory address (e.g., the offset address is used as the data pattern).
    - Starting at Offset Below: The **Print 1K DWORD** button causes the display, in the log window, of 1024 DWORD buffer locations starting at the buffer offset specified in the Offset Address text box.
    - Offset Address (Max = `0xFF000`): The user specifies the lower boundary of the 1024 DWORD blocks to be displayed in the log window. The maximum offset address is `0xFF000`, which is used to display the 1024 DWORDs just below the top of the 1 MB buffer. This text box accepts up to five hexadecimal digits. If non-hexadecimal characters are entered, an error is reported in the log window and the user is requested to try again. The offset address values must be on DWORD boundaries (divisible by 4). If a supported offset value of less than five characters is entered, the text box is auto-filled with leading `0`s. If the user presses the Enter key while the data entry cursor is in the offset address text box, the contents of 1K DWORDs are displayed in the log window. This is an alternative to clicking on the **Print 1K DWORD** button to display the desired data.
- Buffer Offsets: This section permits the selection of memory offsets in the 0 to 64K range in powers of 2 (0, 128, 256, 512, 1K, 2K, 4K, etc.,) for the ML555 DDR2 memory and the host memory buffers.
    - Read DMA (direction): In the read DMA direction, the host PC memory buffer is the data source, and the ML555 DDR2 is the destination memory buffer.
    - Write DMA (direction): In the write DMA direction, the ML555 DDR2 memory buffer is the data source, and the Host PC is the destination memory buffer.
- ML555 Activity Log: This text box displays DMA transaction results and various messages based upon user interaction with the GUI controls.
- **Compare Buffer**: This button checks the 1 MB host memory buffer contents against one of two preset data patterns, 0's or incrementing pattern, depending on which pattern selection radio button is checked.
- **Display RegFile**: This button displays the contents of the ML555 DMA control register file.
- **Reset to Defaults**: This button causes the DMA parameters to be reset to their defaults:
    - Read and write DMA transfer sizes = 128 bytes.
    - Read and write DMA number of transfers = 1.
    - Host memory buffer address offsets = 0. The host buffer address is reset to the originally allocated address shown in the Base Address text box.
- **Clear**: This button clears the ML555 Activity Log window.
- **Print**: This button initiates the process of printing a hard copy of the log window. A Print Preview dialog box is opened to allow preview of the information to be printed, and upon closing this dialog, a general Print dialog box opens, permitting selection of the destination printer.

Understanding PCI Express Transaction Layer Throughput

This section discusses some of the system-level considerations involved in understanding the performance of a PCI Express link, primarily focused on the transaction layer interface, which can be easily monitored inside the FPGA using the ChipScope™ Pro tool [Ref 12]. Information is exchanged between the upstream- and downstream-facing ports through serialized packets. The PCI Express communications model has an application layer, transaction layer, data link layer, and physical layer, as shown in Figure 32. Each layer appends information to the packet to ensure end-to-end data integrity and flow control. The PCI Express Base Specification contains a detailed explanation of the layers and functions provided by each layer. Transactions between the devices are carried using requests and completions. Completions are used to return read data or acknowledge configuration transactions. Completions are associated with corresponding requests by the value of the transaction identification field (tag) in the packet header. Performance can be measured at the physical layer using a protocol analyzer. Inside the FPGA, performance of the transaction layer can be measured using counters.

The clock frequency for the user interface of the Endpoint Block Plus wrapper for PCI Express is selectable at the time the core is generated. For an 8-lane design, the user clock is 250 MHz. The LocalLink interface is always 64 bits, so the maximum throughput is 2000 MB/s for 8-lane designs. Transaction layer packet overhead characters, DDR2 memory read latency, datapath throttling on transmit and receive transaction layer local interfaces (on both sides of the interface), availability of buffer credits, size of packet buffers (on both sides of the interface), round-trip system read latency, and system memory buffer performance and availability must be subtracted from this maximum throughput before making an assessment of typical system-level throughput available in a PC system environment. Other factors, outside the scope of this application note, must be considered in the lower layer protocol layers that are not visible from the transaction layer interface that can be monitored inside the FPGA design.

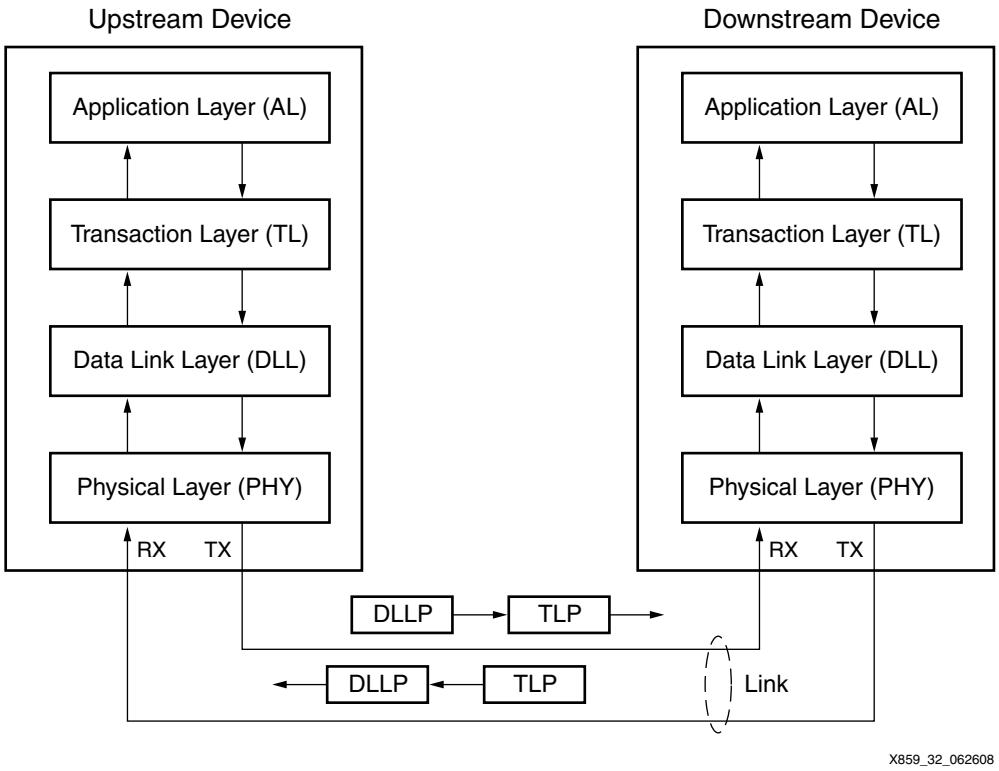


Figure 32: **PCI Express Protocol Layers**

Transaction Layer Packet Header

The application layer creates transaction layer packets with headers that are either 3 double word (3DW) or 4 double word (4DW) (see Figure 12 and Figure 13) depending upon 32-bit or

64-bit address routing. Sixty-four-bit address routing requires one additional double word (DW) of information to be exchanged in the packet. For large data transfers with small maximum payload sizes, the TLP header information must be updated and transmitted in sequential packets. The information in the TLP header format field (byte 0, bit 5) defines the address routing size as follows:

- Logic 0: 32-bit address routing that requires a 3DW header
- Logic 1: 64-bit address routing that requires a 4DW header

Transaction Layer Packet Maximum Payload and Maximum Read Request Size

Device-to-device communication and the resultant throughput is affected by the size and number of packets exchanged between the upstream and downstream ports. The maximum permitted TLP payload is specified as 4 KB. System firmware configures the maximum size of TLP packets exchanged between upstream and downstream devices. In the ASUS P5B-VM, the typical memory read completion packet is 64 bytes long and the maximum posted memory write packet is 128 bytes, even though the Endpoint could be capable of handling larger packets. PCI Express configuration space registers inform the upstream and downstream ports of the device capabilities and the device control settings:

- Maximum Payload Size Capability: The Device Capabilities register defines the maximum payload size that the device can support for TLP packets. The Endpoint Block Plus wrapper supports a maximum TLP packet size of 512 bytes. This parameter only tells the system firmware the maximum payload that the Endpoint is capable of generating. The device control register determines the maximum payload size for DMA write operations.
- Maximum Read Request Size: The Device Control register defines the maximum read request size for a packet. Typically, the maximum read request size is configured to the value of the device's maximum payload size in the device capabilities register. The reference design is capable of handling 512-byte TLP packets. If the firmware configures the maximum read request to be 512 bytes to generate a 4 KB DMA read request, the Endpoint must transmit eight memory read request packets. Transmission of multiple read request packets negatively impacts the Endpoint transmit channel throughput in a full duplex design.
- Maximum Payload Size: The Device Control register also defines the maximum payload size for the port. This register is configured by the system firmware and is not under hardware control. Typically, the Endpoint is configured with a maximum payload size of 128 bytes. This impacts the posted write packets as multiple packets must be transmitted to complete DMA write operations, as shown in Table 11. To generate a 4 KB DMA write, the Endpoint must transmit 32 packets. In a Windows system, memory read completion packets are typically transmitted with 64-byte payloads even though the device control register is configured with 128-byte maximum payloads. For a DMA read operation, the Endpoint receives 64 completion packets with data for a 4 KB DMA read. DMA writes incur half the TLP header packet overhead as DMA reads for a given DMA transfer size.

For memory write and memory read DMA transactions, the effective throughput of the transaction layer interface is reduced because the transaction layer interface must break large data transfers into multiple packets based upon the device configuration specification established by the system firmware. For ease of analysis, it is assumed that both ends of the interface have infinite buffer credits available to receive packets. Table 11 demonstrates the effect of maximum payload size for various sizes of DMA write transfers. Each packet contains a TLP header that must be transmitted along with the data payload. A 4 KB transfer using a 128-byte maximum payload requires a TLP header to be transmitted 32 times, but only once with a 4 KB maximum payload. Table 12 shows the maximum theoretical throughput difference and TLP header packet byte overhead at the TLP layer for various payload sizes.

Table 11: **Number of DMA Write Packets Versus Maximum Payload Size**

DMA Transfer Size (KB)	Number of Posted Memory Write Packets to Transmit for DMA Write with Maximum TLP Payload Size (Bytes)			
	128(1)	256	512	4096(2)
0.125	1	1	1	1
0.25	2	1	1	1
0.5	4	2	1	1
1	8	4	2	1
2	16	8	4	1
4	32	16	8	1
8	64	32	16	2
16	128	64	32	4
32	256	128	64	8
64	512	256	128	16
128	1024	512	256	32
256	2048	1024	512	64
512	4096	2048	1024	128
1024	8192	4096	2048	256

Notes:

1. Double the number of TLP packets for DMA read completion packets due to the typical 64-byte completion packet payload size in a Windows-based PC system.
2. Maximum payload size of 4 KB is specified by the PCI Express Base Specification.

As an example of the throughput degradation caused by transmission of multiple write packets, consider a posted memory write transaction. The Endpoint Block Plus wrapper for PCI Express supports a maximum TLP payload of 512 bytes, whereas system firmware can limit the TLP payload to 128 bytes using the Device Control register. Assuming that the system permits maximum write payloads of 128 bytes, the Endpoint must transmit 32 packets, each with 128 bytes of payload, to complete a 4 KB DMA write. The transaction layer LocalLink interface is 64 bits wide or 8 bytes per clock. Table 12 demonstrates the impact of multiple TLP packets on transaction layer data throughput for a 4 KB posted memory write DMA operation for 8-lane Endpoint designs and for different TLP payload sizes.

Table 12: **Theoretical Maximum TLP Data Throughput as a Function of Payload Size for a 4 KB Transfer**

TLP Payload Size (Bytes)	Number of TLP Packets to Complete 4 KB Transfer	TLP Header Overhead (Bytes) Number of TLPs X 16 Bytes(1)	Theoretical Maximum TLP Data Throughput(2,3)
64(4)	64	1024	80.00%
128(5)	32	512	88.89%
256(6)	16	256	94.12%
512	8	128	96.97%
1024	4	64	98.46%
2048	2	32	99.22%

Table 12: **Theoretical Maximum TLP Data Throughput as a Function of Payload Size for a 4 KB Transfer *(Cont'd)***

TLP Payload Size (Bytes)	Number of TLP Packets to Complete 4 KB Transfer	TLP Header Overhead (Bytes) Number of TLPs X 16 Bytes(1)	Theoretical Maximum TLP Data Throughput(2,3)
4096	1	16	99.61%

Notes:

1. The TRN data interface is 8 bytes wide with a remainder function active during the end-of-frame assertion. For TLPs with even-numbered DWORD payload sizes, the header overhead can be assumed to always be 16 bytes for both 3DW and 4DW headers sizes; i.e., for 3DW headers the remainder is included as header overhead.
   - 3DW = 12-byte header + 4-byte remainder = 16-byte overhead
   - 4DW = 16-byte header = 16-byte overhead
2. Throughput % = 100 x DMA Transfer Size / (DMA Transfer Size + TLP Header Overhead)
3. Real throughput is lower due to Data Link and Physical Layer Packet overhead and finite Flow Control Credits.
4. Payload sizes of 64 bytes are common because most chipsets and systems utilize a read completion boundary of 64 bytes.
5. The ASUS P5B-VM system, used for hardware testing of this reference design, utilizes a 128-byte maximum payload size.
6. The Dell Precision 690 workstation uses an Intel 5000X chipset supporting a 256-byte maximum payload size.

DDR2 Read Latency

SODIMM memory must be accessed before the TLP posted write packets can begin transmission. The DDR2 refresh interval is 7.8 µs. The MIG-based DDR2 design automatically handles DRAM refresh cycles by prioritizing refresh above read or write accesses. In the event a DMA write operation just starts to access DDR2 memory and the DRAM refresh timer expires, the data flow from the memory controller to the transmit engine stalls the pipeline. The DDR2 read commands occur after the refresh cycle completes. However, the memory active command and read latency need to occur, and this impacts the performance of the first packet. After the first transmit packet has been fetched from memory, the refresh interval can occur anytime in future packet accesses without impacting performance.

The *MIG User Guide* [Ref 7] provides estimates for DDR2 controller read latencies. The reference design has additional latencies on top of the DDR2 read latency, including the time to get through an internal FIFO to convert the 16-byte-wide memory interface to an 8-byte interface for the Endpoint block. The DDR2 interface returns data faster than the DMA engine can transfer packets on the PCI Express link. Thus, data is stored internally and pulled out of the FIFO when the transmit engine is ready to send data. DDR2 read latency, approximately 220 ns, only impacts the time to get the first write DMA packet transmitted. After the write operation has started, the DDR2 memory continues to read data, and the transmit engine never waits for data from the memory controller.

The *Bus Master DMA Reference Design for the Xilinx Endpoint Block Plus Core for PCI Express Designs* application note [Ref 13] provides an example bus mastering DMA endpoint reference design, without a backend DDR2 interface, that can be utilized to test the throughput of a x1, x4, or x8 lane endpoint transaction layer interface with various TLP payload sizes.

Round-Trip System Read Latency

The round-trip system read latency is very much system and chip-set dependant, difficult to predict, and impacts the DMA read throughput. Items in this system-specific parameter include:

- Time to create multiple memory read requests in accordance with the maximum read request size configuration space parameter.
- Time to transmit the TLP packets on the transmit TRN interface, including any datapath throttling.

- Time to go from endpoint transmit TRN interface out the transceiver and through the protocol layer stack at the upstream port.
- Any data link and physical layer packet overhead.
- Time to arbitrate for the system memory buffer, including time waiting for DRAM refresh cycles to complete or other processor memory accesses to complete. Time to read the system memory buffer locations and then create TLP completion with data packets in accordance with the read completion payload size of the system.
- Time to transmit multiple TLP completion with data packets on the upstream port.
- Time to transition through the downstream port protocol layers, finally arriving on the receiver transaction layer interface with a start of frame assertion.

Transaction Layer Source and Destination Datapath Throttling

The transmit transaction LocalLink interface permits the core to insert wait states to the user application logic by deasserting the *transmit destination ready* port, trn_tdst_rdy_n, which could occur if an upstream component has exhausted all of its buffer credits. When deasserted, the user logic must wait until the core is ready to accept data before continuing to the next data beat. User application logic deasserts the transmit source ready port, trn_tsrc_rdy_n, when it needs to throttle the datapath, which occurs when data is unavailable because DRAM refresh cycles are in process. Insertion of wait states on the transmit datapath impacts the DMA write performance, as shown in Table 13. It is difficult to predict if or when wait states are asserted on the LocalLink interface. This example demonstrates the effect that wait states can have on the transaction layer throughput.

The receiver transaction LocalLink interface provides source and destination datapath throttling controls: receiver source ready (trn_rsrc_rdy_n) and receiver destination ready ports (trn_rdst_rdy_n). The user application logic never throttles the receive datapath using trn_rdst_rdy_n because it can accommodate much higher data rates using a DDR interface. It is important to note that because completion streaming is used and the endpoint device advertises infinite completion credits, the user should not throttle the receive interface using trn_rdst_rdy_n.

Table 13: **Impact of Wait States on 4096-Byte Posted Write Performance**

Number of 128-Byte Packets to Transmit a 4 KB Posted Write	Number of Wait States on TRN LocalLink Interface	Per-Packet Transmission Time for 8-Lane PCIe (ns)	Time to Complete Transfer (1,2) (ns)	MB/s over 8-Lane PCIe Link
32	0	72	2494	1642
32	1	76	2622	1562
32	2	80	2750	1489
32	3	84	2878	1423
32	4	88	3006	1362

Notes:

1. Includes 220 ns time from Write_DMA_Start assertion to first data available from DDR2 memory.
2. Assumes that the same number of wait states are inserted on each of the 32 packets transmitted.

ML555 DMA Throughput in a System Unit

The throughput of the reference design as measured on a Dell 690 system and an ASUS motherboard is shown in Table 14, Table 15, and Table 16. Throughput is measured from the Start_DMA control assertion to the end of packet of the last memory write packet or the last completion-with-data packet received on a read cycle. For DMA write transfers, the DDR2 interface is known to have an approximate 220 ns read latency. For DMA read transfers, the system read latency is approximately 840 ns in both the Dell and ASUS systems for the x8 configurations.

Table 14: **ML555 DMA Throughput of 8-Lane Endpoint Block DMA Initiator Design In Dell 690 System(5)**

DMA Transfer Size (KB)(1)	Half Duplex		Full Duplex	
	Memory Read DMA (MB/s)(2,3)	Memory Write DMA (MB/s)(2,4)	Memory Read DMA (MB/s)(2,3)	Memory Write DMA (MB/s)(2,4)
0.125	136	405	136	385
0.25	245	673	245	673
0.5	408	969	408	920
1	622	1261	506	1261
2	833	1528	712	1471
4	1030	1695	970	1575
8	1159	1782	991	1644
16	1238	1816	1137	1708
32	1280	1830	1064	1739
64	1303	1800	1023	1747
128	1314	1785	1032	1735
256	1320	1778	1028	1729
512	1323	1775	1029	1726
1024	1325	1773	1029	1725

Notes:

1. DMA transfer size is defined by parameter DMAWXS or DMARXS in Table 3.
2. These numbers are typical for the Dell 690 and can vary.
3. Throughput numbers include approximately: 840 ns of round-trip system read latency; TLP maximum read request of 512 bytes; and TLP read completion payload of 64 bytes. Transaction layer wait states can vary. See Figure 34 for an example DMA Memory Read on the transaction layer interface inside the FPGA.
4. Throughput numbers include approximately 220 ns of startup latency. Transaction layer wait states can vary. See Figure 33 for an example DMA Memory Write on the transaction layer interface inside the FPGA.
5. Parameters:
   Link Width: x8
   Maximum Payload Size: 256 bytes
   Maximum Read Request Size: 512 bytes
   Root Complex Read Completion Boundary: 64 bytes

Table 15: **ML555 DMA Throughput of 8-Lane Endpoint Block DMA Initiator Design In Dell 690 System Down-Trained to Four Lanes(5)**

DMA Transfer Size (KB)(1)	Half Duplex		Full Duplex	
	Memory Read DMA (MB/s)(2,3)	Memory Write DMA (MB/s)(2,4)	Memory Read DMA (MB/s)(2,3)	Memory Write DMA (MB/s)(2,4)
0.125	125	410	125	390
0.25	216	680	215	673
0.5	335	992	329	1000

Table 15: **ML555 DMA Throughput of 8-Lane Endpoint Block DMA Initiator Design In Dell 690 System Down-Trained to Four Lanes[5] *(Cont'd)***

DMA Transfer Size (KB)[1]	Half Duplex		Full Duplex	
	Memory Read DMA (MB/s)[2,3]	Memory Write DMA (MB/s)[2,4]	Memory Read DMA (MB/s)[2,3]	Memory Write DMA (MB/s)[2,4]
1	462	1299	382	1207
2	570	1537	464	1475
4	646	1486	563	1276
8	692	1128	555	929
16	718	1001	553	865
32	732	954	546	816
64	739	931	553	799
128	742	920	554	787
256	744	915	557	778
512	745	912	557	775
1024	745	911	558	777

Notes:

1. DMA transfer size is defined by parameter DMAWXS or DMARXS in Table 3.
2. These numbers are typical for the Dell 690 and can vary.
3. Throughput numbers include approximately: 890 ns of round-trip system read latency; TLP maximum read request of 512 bytes; and TLP read completion payload of 64 bytes. Transaction layer wait states can vary. See Figure 34 for an example DMA Memory Read on the transaction layer interface inside the FPGA.
4. Throughput numbers include approximately 220 ns of startup latency. TLP maximum payload size is 128 bytes. Transaction layer wait states can vary. See Figure 33 for an example DMA Memory Write on the transaction layer interface inside the FPGA.
5. Parameters:
   Link Width: x4
   Maximum Payload Size: 256 bytes
   Maximum Read Request Size: 512 bytes
   Root Complex Read Completion Boundary: 64 bytes

Table 16: **ML555 DMA Throughput of 8-Lane Endpoint Block DMA Initiator Design in ASUS P5B-VM System[6]**

DMA Transfer Size (KB)[1]	Half Duplex		Full Duplex	
	Memory Read DMA (MB/s)[2,3]	Memory Write DMA (MB/s)[2,4]	Memory Read DMA (MB/s)[2,3]	Memory Write DMA (MB/s)[2,4]
0.125	135	390	133	376
0.25	246	646	229	646
0.5	416	948	403	934
1	644	1236	563	1190
2	906	1458	709	1242
4	1121	1294	947	895
8	1284	1169	954	827
16	1378	1113	958	824
32	1430	1077	906	829
64	1458	1078	870	870
128	1476	1076	872	862
256	1484	1075	870	863

Table 16: **ML555 DMA Throughput of 8-Lane Endpoint Block DMA Initiator Design in ASUS P5B-VM System(6)**

DMA Transfer Size (KB)(1)	Half Duplex		Full Duplex	
	Memory Read DMA (MB/s)(2,3)	Memory Write DMA (MB/s)(2,4)	Memory Read DMA (MB/s)(2,3)	Memory Write DMA (MB/s)(2,4)
512	1488	1074	872	865
1024	1491	1074	877	870

Notes:

1. DMA transfer size is defined by parameter DMAWXS or DMARXS in Table 3.
2. These numbers are typical for the ASUS P5B-VM and can vary.
3. Throughput numbers include approximately: 840 ns of round-trip system read latency; TLP maximum read request of 512 bytes; and TLP read completion payload of 64 bytes. Transaction layer wait states can vary. See Figure 34 for an example DMA Memory Read on the transaction layer interface inside the FPGA.
4. Throughput numbers include approximately 220 ns of ML555 DDR2 memory read latency. Transaction layer wait states can vary. See Figure 33 for an example DMA Memory Write on the transaction layer interface inside the FPGA.
5. Parameters:
   Link Width: x8
   Maximum Payload Size: 128 bytes
   Maximum Read Request Size: 512 bytes
   Root Complex Read Completion Boundary: 64 bytes

Figure 33 and Figure 34 show how DMA performance is measured for 4 KB write and read DMA transfers, respectively. Figure 33 shows the DDR2 read latency time as well as wait states inserted due to the upstream component exhausting its buffer credits. Figure 34 shows the system turnaround latency included in the read DMA throughput calculation.

For DMA read, system read latency has a large impact on throughput so larger transfer sizes are more efficient. For DMA writes, the system memory can become a bottleneck with performance decline going from 2 KB to 4 KB as datapath throttling begins with two large wait state periods in Figure 33.

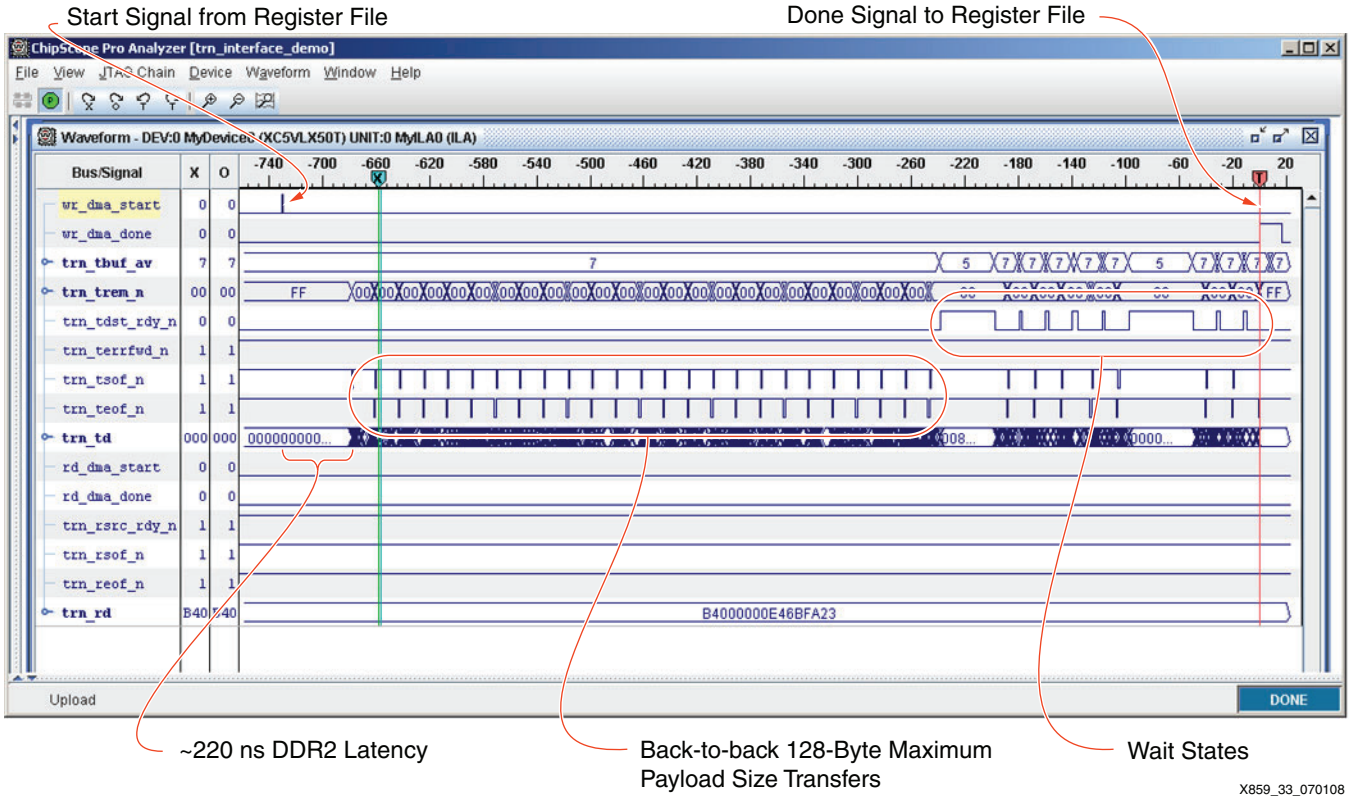


Figure 33: **4 KB Write Example**

Memory write throughput on the transaction interface, as measured in the ASUS and Dell systems includes:

- Approximately 220 ns to read data from the DDR2 memory in the ML555 board.
- Time to create and transfer multiple posted write packets on the TRN interface in accordance with the maximum payload size configuration space setting defined by the host processor.
- Any wait states inserted on the endpoint transmitter TRN interface.
- TLP packet header overhead, as shown in Table 12, for TLP payload size.

In the write example shown in Figure 33, after the 220 ns DDR2 read latency time, the endpoint transmit TRN interface transmits at the maximum theoretical throughput rate (no wait states) for twenty-four consecutive 128-byte TLP packets. After approximately 3072 data bytes have been transferred, the datapath is throttled back by the deassertion of the trn_tdst_rdy_n signal for two significant periods of time. The system is saturated and cannot accept any more data. For this particular PC system, the maximum throughput is achieved for 2 KB posted writes, as shown in Table 16.

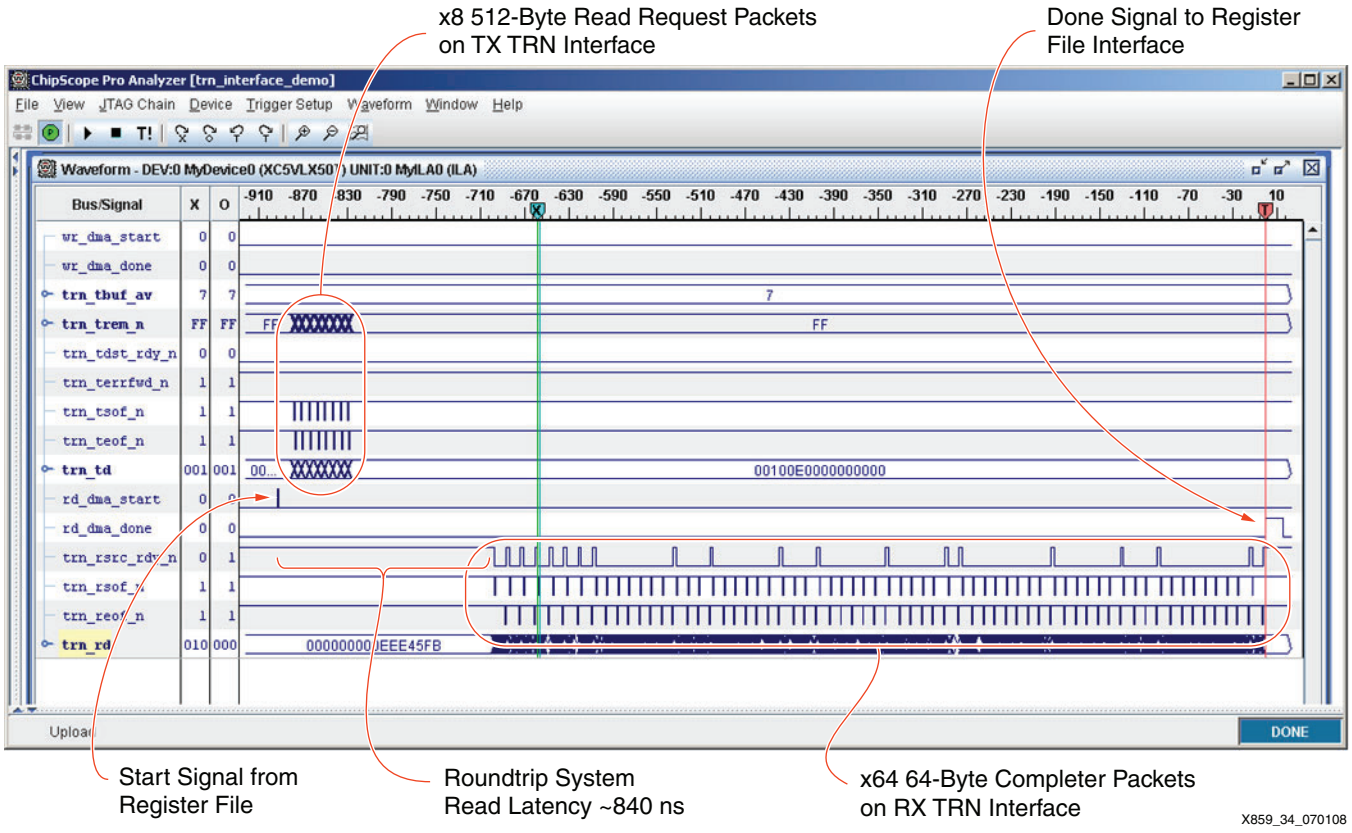


Figure 34: **4 KB Read Example**

Memory read throughput on the transaction interface, as measured in the ASUS and Dell systems includes:

- Time to create multiple memory read request packets in accordance with the maximum read request configuration space setting defined by the host processor
- Time to transfer multiple memory read requests on the transmit TRN interface, including any datapath throttling
- Round-trip physical layer transmitter and receiver packet transfer times for crossing PHY layers of the two devices

- Time to arbitrate and access the system memory buffer and create multiple 64-byte completion with data TLP packets
- Time to receive multiple completer with data packets, including any datapath throttling on the receiver TRN interface

In the read example shown in Figure 34, the creation of multiple 512-byte memory read requests can be seen on the transmit TRN interface. The time from the start of DMA transfer to the start of the first completion with data is called the round-trip system read latency and is approximately 840 ns, but can vary. After the TLP completion packets with data start arriving on the receiver TRN interface, very few wait states are requested, as can be seen from the trn_rsrc_rdy_n signal. Looking at Table 16 and Figure 34, the system throughput has not saturated for the 4 KB read example.

Reference Design

The reference design for this application note can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=109265

The reference design ZIP archive contains a `readme.txt` file that describes the individual files and their usage in more detail. This `readme.txt` file also contains important notes that the user should adhere to when simulating or implementing the design.

The reference design matrix is shown in Table 17.

Table 17: **Reference Design Matrix**

General	
Developer Name	Xilinx
Target Devices (stepping level, ES, production, speed grades)	Virtex-5 FPGA LX50T production
Source Code Provided	Yes
Source Code Format	Verilog
Design Uses Code/IP from an Existing Reference Design/Application Note, 3rd Party, or CORE Generator software	Yes
Simulation	
Functional Simulation Performed	Yes
Timing Simulation Performed	No
Testbench Used for Functional Simulations Provided	Yes
Testbench Format	Verilog
Simulator Software Used/Version (e.g., ISE software, Mentor, Cadence, other)	ModelSim SE 6.3c
SPICE/IBIS Simulations	No
Implementation	
Synthesis Software Tools Used/Version	XST
Implementation Software Tools Used/Versions	ISE 10.1.02
Static Timing Analysis Performed	Yes
Hardware Verification	
Hardware Verified	Yes
Hardware Platform Used for Verification	ML555 + ASUS P5B-VM PC ML555 + Dell Precision 690

Conclusion

This reference design utilizes the CORE Generator software to create an 8-lane LogiCORE Endpoint Block Plus Wrapper for a PCI Express design. This is done to initiate DMA writes and reads to a system buffer memory located in a system unit. The ML555 development kit is used to validate the design and uses the onboard 256 MB DDR2 SODIMM as a backend memory for storing DMA data. High-throughput data rates are achieved using the 8-lane Endpoint Block Plus with a user application running at 250 MHz to match the line rate of the PCI Express bus. A fully synthesized and simulated reference design is provided as an example of how to design with the Endpoint Block Plus LogiCORE system at line rate speeds.

References

The following references were used in this application note:

1. DS551, *Endpoint Block Plus for PCI Express Product Specification*.
2. UG341, *LogiCORE Endpoint Block Plus for PCI Express User Guide*.
3. UG343, *LogiCORE Endpoint Block Plus for PCI Express Getting Started Guide*.
4. PCI Express Base Specification, Version 1.1, PCI-SIG.
5. XAPP858, *High-Performance DDR2 SDRAM Interface in Virtex-5 Devices*.
6. WP260, *Memory Interfaces Made Easy with Xilinx FPGAs and the Memory Interface Generator*.
7. UG086, *Xilinx Memory Interface Generator (MIG) User Guide*.
8. DS202, *Virtex-5 FPGA Data Sheet: DC and Switching Characteristics*.
9. UG201, *Virtex-5 FPGA ML555 Development Kit for PCI and PCI Express Designs User Guide*.
10. UG197, *Virtex-5 FPGA Integrated Endpoint Block for PCI Express Designs User Guide*.
11. WinDriver PCI/ISA/CardBus v9.20 User's Manual, Jungo Ltd. http://www.jungo.com/st/support/documentation/windriver/920/wdpci_man.pdf.
12. XAPP1002, *Using ChipScope Pro to Debug Endpoint Block Plus Wrapper, Endpoint, and Endpoint PIPE Designs for PCI Express*.
13. XAPP1052, *Bus Master DMA Reference Design for the Xilinx Endpoint Block Plus Core for PCI Express Designs*
14. UG347, *ML505/ML506/ML507 Evaluation Platform User Guide*

Additional Resources

Additional information can be found in these resources.

1. UG190, *Virtex-5 FPGA User Guide*.
2. UG191, *Virtex-5 FPGA Configuration User Guide*.
3. UG196, *Virtex-5 FPGA RocketIO GTP Transceiver User Guide*.
4. UG198, *Virtex-5 FPGA RocketIO GTX Transceiver User Guide*.
5. Budruk, Ravi, Don Anderson, and Tom Shanley. 2003. *PCI Express System Architecture*. Addison-Wesley Professional.

Appendix: Porting the Reference Design to the ML505 Board

The reference design is ported to the ML505 board and is provided in the accompanying `xapp859.zip` file. This section details the differences between the ML555 and ML505 reference designs and provides the necessary steps required to run the reference design on a PCI Express system, such as the ASUS P5B-VM.

Table 18 lists the key differences between the ML505 and ML555 reference designs.

Table 18: **Differences between ML555 and ML505 Reference Designs**

Parameter	ML505	ML555
Lane Width	x1	x8
PCI Express Lane 0 GTP Transceiver Assignment	GTP_X0Y1	GTP_X0Y2
Transaction Interface (TRN) Clock Frequency	62.5 MHz	250 MHz
PCI Express Configuration Register Revision ID(1)	`0x01`	`0x00`

Table 18: **Differences between ML555 and ML505 Reference Designs *(Cont'd)***

Parameter	ML505	ML555
DDR2 SDRAM Clock Source (2)	On-board, fixed 200 MHz oscillator	On-board, programmable clock source
PCIE_PERST_B Pin Source	On-board dip switch	PCI Express edge connector
Status LED Polarity	Active-High	Active-Low

Notes:

1. The user application GUI uses the PCI Express configuration register Revision ID to determine which board type is present in the system. The application uses this information to determine the TRN clock frequency, which it then uses to calculate the data throughput.
2. The ML505 board uses a fixed, on-board, 200 MHz oscillator for the DDR2 SDRAM memory. This clock is shared with the IDELAY_CTRL element. The ML555 board uses a programmable clock for the DDR2 SDRAM memory while the IDELAY_CTRL has its own 200 MHz clock source.

Other differences between the two designs include the pin assignments of the DDR2 SDRAM, status LED outputs, and the PCI Express reset pin PCIE_PERST_B. The respective UCF files can be examined to determine the differences in pin assignment.

The instructions in the "LogiCORE Endpoint Block Plus Wrapper for PCI Express Designs" section should be followed to rebuild the Endpoint Block Plus core using the CORE Generator software for the ML505 board. A few parameter differences are noted here:

- Number of Lanes: x1
- Interface Frequency: 62.5 default
- Revision ID: `0x01`

ML505 Board Setup

The *ML505/ML506/ML507 Evaluation Platform User Guide* [Ref 14] should be consulted before installing the ML505 board into the system unit. The items shown in Figure 35 should be verified before installing the ML505 board in the PC system and applying power to the system unit and ML505 board. The steps to install the ML505 board are:

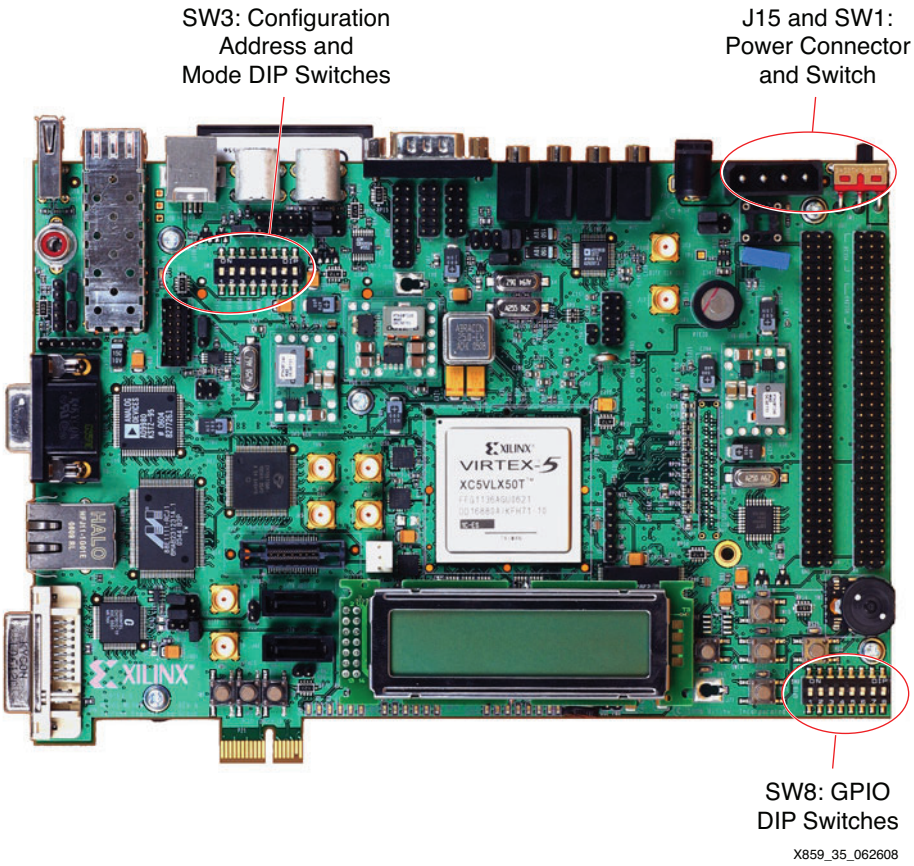


Figure 35: **ML505 Board Setup**

1. Connect a spare PC-type disk drive connector from the host PC system to J15.
2. Set SW1 to the ON position (slide to the right), as shown in Figure 36.

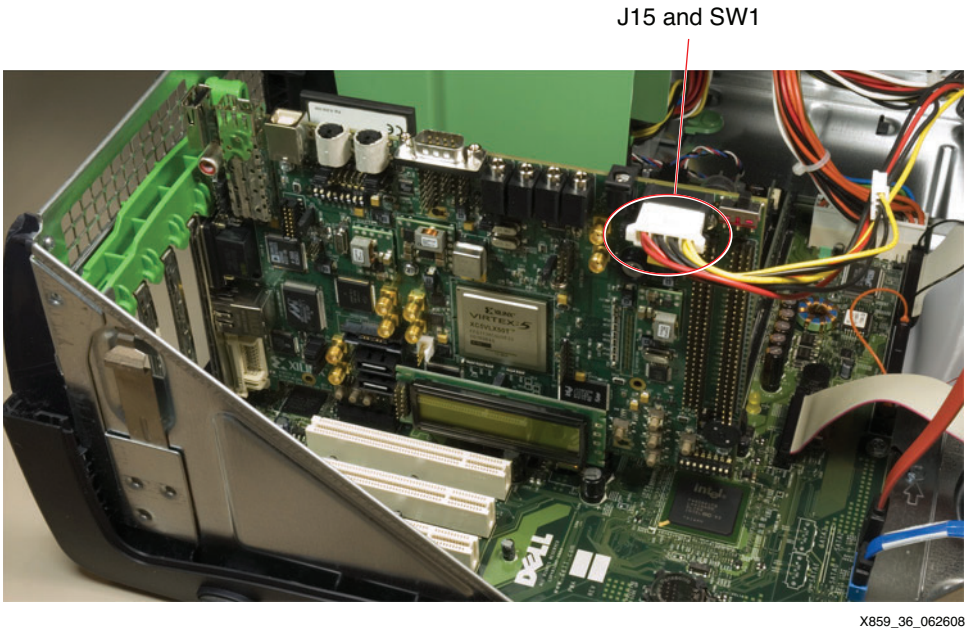


Figure 36: **J15 and SW1 Switch**

3. Set the Configuration Address and Mode DIP switches (SW3) to `01010000` (`1` = ON), as shown in Figure 37.

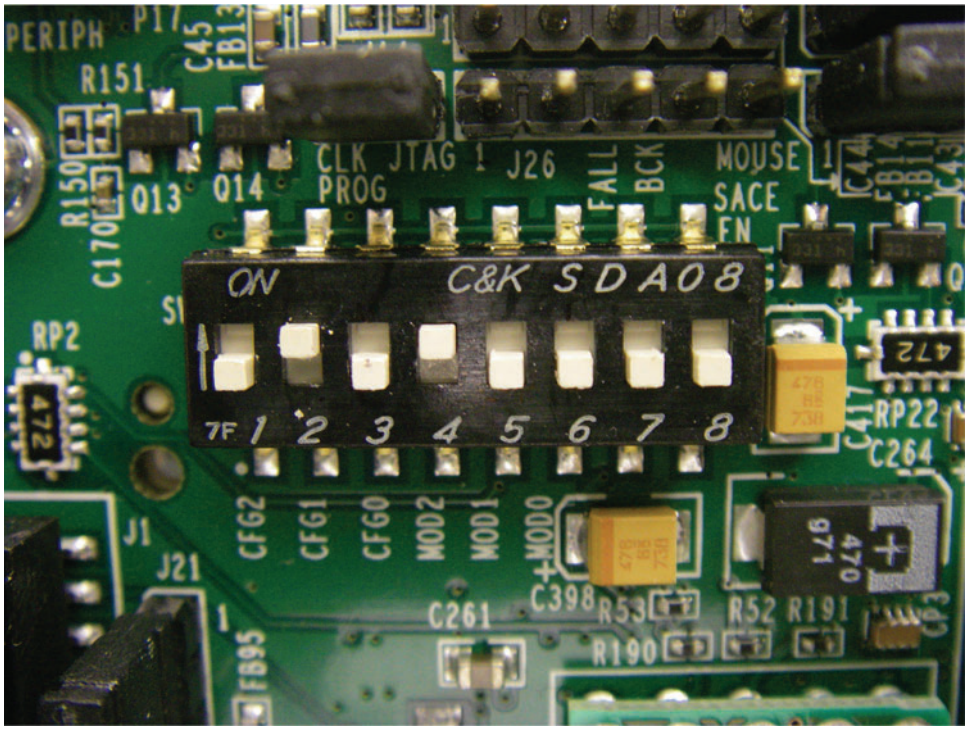


Figure 37: **Configuration Address and Mode DIP Switches**

4. Set the GPIO DIP switches (SW8), and bits 7 and 8 to `1` (`1` = ON), as shown in Figure 38.

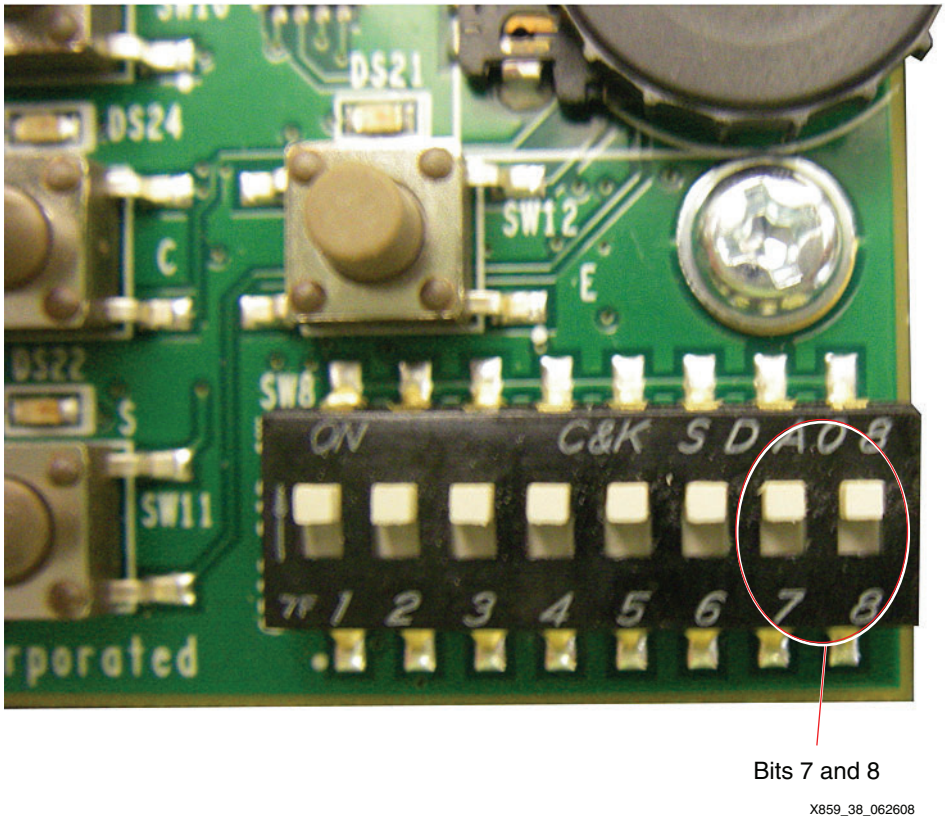


Figure 38: **Bits 7 and 8**

Proper operation of this reference design requires headers on the ML505 board to have shunts installed in accordance with Table 19. These shunts are provided with the ML505 board. In most cases, Table 19 reflects the default configuration of the ML505 board, as built. Optional headers are listed for completeness. Prior to installing the ML505 board on the PC, the proper configuration of the board headers must be verified. Any header on the ML505 board not listed in Table 19 should be left as an open circuit.

Table 19: **ML505 Shunt Installation Requirements for XAPP859 Operation**

ML505 Board Reference Designator	Connectivity for XAPP859 PCI Express Application	Comments	Function Provided
SW3[1:8]	`01010000`(2)	Required	Configuration Address and Mode DIP switches: Config address 2, Master SelectMAP mode, platform flash
SW8[1:8]	`11111111`(2)	Required	General Purpose Input/Output (GPIO) DIP switches: Bits 7 and 8 must be set to `11`
J21	1 to 2	Required	Bypass Xilinx Generic Interface (XGI) expansion header JTAG
SW6 [1:8]	`11001010`(2)	Optional(1)	Clock source and frequency setting for clock synthesizer device ICS843001-21
J9	1 to 2 3 to 4	Optional(1)	System monitor header

Table 19: **ML505 Shunt Installation Requirements for XAPP859 Operation *(Cont'd)***

ML505 Board Reference Designator	Connectivity for XAPP859 PCI Express Application	Comments	Function Provided
J14	1 to 2	Optional(1)	IIC/JTAG_B pin
J17	2 to 3	Optional(1)	FPGA AVDD select
J20	1 to 3 2 to 4	Optional(1)	Bank 11/13 expansion connector voltage select
J18, J19	Open circuit	Optional(1)	System ACE™ interface failsafe mode
J22, J23	1 to 2	Optional(1)	Ethernet mode select
J24	Open circuit	Optional(1)	Ethernet mode select
J33	Open circuit	Optional(1)	USB abort
J54	1 to 2	Optional(1)	Clock multiplexer select
J62, J63	1 to 2	Optional(1)	Serial port select
J56	1 to 2	Optional(1)	Serial ATA (SATA) clock frequency select
J81	1 to 2	Optional(1)	Small form-factor pluggable (SFP) full bandwidth (BW)
J82	1 to 2	Optional(1)	SFP transmitter (TX) enable

Notes:

1. Optional shunt installation. Does not affect operation of this reference design.
2. `1` = ON, `0` = OFF.

The reference design ZIP file includes the FPGA and platform flash configuration files `xapp859_ml505.bit` and `xapp859_ml505.mcs`. The BIT and MCS files are manually loaded into the FPGA and platform flash, respectively, using the JTAG interface of the ML505 board. A script is provided with the reference design which executes iMPACT in batch mode to program the platform flash. See the `readme.txt` file provided in `xapp859.zip` for more information.

The preferred method to run the reference design is to load the platform flash image of the ML505 board, and cycle the power off and on. After the image is loaded into the platform flash, the reference design is automatically loaded at power on, and the design runs in the system environment without having to restart the computer. The PC system firmware enumerates the ML505 board and configures the add-in card adapter as part of the system initialization process.

The *ML505/ML506/ML507 Evaluation Platform User Guide* [Ref 14] contains a detailed board and FPGA configuration section that should be consulted for information about how to load the BIT and MCS files on the ML505 board. When running BITGEN to produce a design image for the ML505 board, a CCLK frequency of 20 MHz must be used, rather than the default 2 MHz. For example:

```
bitgen -g ConfigRate:20 <infile>
```

This concludes the setup for the ML505 board. Other procedures for running the reference design such as "Installing the ML555 Jungo Device Driver on Windows," page 39 and "Running the DMA Demonstration User Application," page 41 are identical to those for the ML555 board and can be found in the respective sections of the application note.

ML505 DMA Throughput

The throughput of the 1-lane Endpoint block design as measured on the ASUS P5B-VM system unit is shown in Table 20. Throughput is measured from the Start_DMA control assertion to the end of packet for the last memory write packet on a write cycle, or the last completion-with-data packet received on a read cycle. For DMA write transfers, there is an approximate start-up latency of 400 ns. This is not due to the DDR2 read latency as is the case with the ML555 board but rather, it is due to the slower 62.5 MHz TRN interface clock that the ML505 board uses. For DMA read transfers, the system read latency is approximately 2000 ns.

Table 20: **ML505 DMA Throughput of 1-Lane Endpoint Block DMA Initiator Design In ASUS P5B-VM System (5)**

DMA Transfer Size (KB)(1)	Half Duplex		Full Duplex	
	Memory Read DMA (MB/s)(2,3)	Memory Write DMA (MB/s)(2,4)	Memory Read DMA (MB/s)(2,3)	Memory Write DMA (MB/s)(2,4)
0.125	50	160	50	156
0.25	79	235	76	235
0.5	110	307	102	307
1	139	363	119	347
2	159	398	127	376
4	172	286	148	256
8	179	245	154	219
16	182	228	154	205
32	184	221	151	200
64	185	218	155	196
128	186	216	155	194
256	186	215	156	194
512	186	215	156	193
1024	186	215	156	193

Notes:

1. DMA transfer size is defined by parameter DMAWXS or DMARXS in Table 3.
2. These numbers are typical for the ASUS P5B-VM and can vary.
3. Throughput numbers include approximately: 2000 ns of round-trip system read latency; TLP maximum read request of 512 bytes; and TLP read completion payload of 64 bytes. Transaction layer wait states can vary. See Figure 34 for an example DMA Memory Read on the transaction layer interface inside the FPGA.
4. Throughput numbers include approximately 400 ns of startup latency. Transaction layer wait states can vary. See Figure 33 for an example DMA Memory Write on the transaction layer interface inside the FPGA.
5. Parameters:
   Link Width: x1
   Maximum Payload Size: 128 bytes
   Maximum Read Request Size: 512 bytes
   Root Complex Read Completion Boundary: 64 bytes

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
04/18/08	1.0	Initial Xilinx release.
07/31/08	1.1	• Updated "Summary," page 1, "Read Request Wrapper," page 20, "Design Implementation," page 28, "Running the Reference Design Demonstration: Quick Start Guide," page 31, "ML555 Board Setup," page 31, "Running the DMA Demonstration User Application," page 41, "DDR2 Read Latency," page 48, "ML555 DMA Throughput in a System Unit," page 50, and "References," page 56. • Updated Table 4, Table 7, Table 8, Table 12, Table 13, Table 16, Table 17, Figure 33, and Figure 34. • Added Table 14 and Table 15. • Added "Installing Microsoft .NET Framework Version 2.0," page 36 and "Appendix: Porting the Reference Design to the ML505 Board," page 56.

Notice of Disclaimer