5-2010

FPGA BASED TIMING MODULE AND OPTICAL COMMUNICATION CARD DESIGN FOR SPALLATION NEUTRON SOURCE

Biswa Singh
Clemson University, biswagourav.singh@gmail.com

FPGA BASED TIMING MODULE AND OPTICAL COMMUNICATION CARD DESIGN FOR SPALLATION NEUTRON SOURCE

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

by
Biswa G. Singh
May 2010

Accepted by:
Melissa Smith, Committee Chair
Walter Ligon
Richard Brooks

ABSTRACT

The Timing Module and Optical Communication Card (OCC) are used for acquisition of neutron event data by the instrument systems at the Spallation Neutron Source (SNS) neutron scattering facility. The instrument systems produce a very large flux of neutrons of varying energies over a short time period through the spallation process. The Timing Module and OCC require high-bandwidth communication to ensure high-speed data movement to the memory in the data collection system without loss of neutron data. The existing implementations use a standard PCI-X bus interface to transfer the data between the cards and the host computer. The data processing on the existing cards is implemented in a Xilinx Virtex-II FPGA. The bandwidth restrictions of the PCI-X bus and the logic constraints of the Virtex-II FPGA have resulted in limited capabilities of the instrument systems. New designs for the timing and communication modules that will improve performance, avoid data loss, and provide for future logic expansion are desired.

In this project, we redesign the Timing Module and OCC moving from a PCI-X to PCI-Express bus interface to improve the data acquisition bandwidth. The new design also uses a Xilinx Virtex-5 FPGA to allow more channels to be processed per card and provide for further expansion. Further, the Virtex-5 device also has an embedded PCI-Express Hard IP core. This internal core simplifies the Printed Circuit Board (PCB) design since there is no external PCI interface chip required and decreases the probability of errors between the PCI interface and user logic design. The Timing Module implements a simple PCI Express read and write for the data transfer. The OCC requires a higher data rate than the Timing Module and therefore uses a more complex bus master direct memory access (DMA) for the endpoint PCI-Express block, which allows for lower CPU utilization and higher performance.

New user logic interfaces were designed to integrate the PCI-Express endpoint with the Timing Module and the OCC logic designs. A single PCB was designed to function as both the Timing Module and OCC. The logic designs were verified by both functional simulation and in-system JTAG signal capture on the new PCB. The results indicate that our design provides efficient data transfer, higher throughput, and scalability, benefitting both modules and meeting design requirements.

DEDICATION

The thesis is dedicated to my family and friends.

ACKNOWLEDGEMENT

The work presented in this thesis would not have been possible without the help and support of many. My deepest gratitude goes to my advisor, Dr. Melissa Smith for her guidance and support that made this work possible. Next, I would like to acknowledge my fellow graduate students who have spent a lot of time discussing the work contained herein. I would especially like to thank Steve Hicks from Oak Ridge National Laboratory for his sincere support.

TABLE OF CONTENTS

Table of Contents (Continued) Page

LIST OF TABLES

LIST OF FIGURES

List of Figures (Continued)

List of Figures (Continued)

CHAPTER ONE

INTRODUCTION

The Spallation Neutron Source (SNS) is a facility at Oak Ridge National Laboratory (ORNL) for neutron-scattering research. Neutron-scattering research has wide spread applications in improving medicines, foods, electronics, automobiles, and avionics. The instrumentation systems at SNS produce neutrons of different energies by a spallation process where high energy protons impact a target. The instrument system's data acquisition system (DAS) [1] is designed for control and collection of neutron-scattering data. The DAS group has developed a number of electronic boards which are capable of collecting data from neutron detector electronics. In addition to detector electronics, the boards are also used to send real-time digital signals via fiber optics. The need for high data rate capabilities necessitated the development of custom hardware boards. Two of the custom boards that accomplish these tasks are the PCI based Timing Module card and optical communication card (OCC).

The Timing Module [2, 3] maintains various registers that control the configuration and operation of the timing cards. The registers are responsible for maintaining the synchronization of the detector electronics with the chopper and accelerator phasing. The Timing Module also provides the master timing pulse to the chopper control system for phasing the chopper disk with the proton on target event. The registers are accessed by the host CPU over a PCI interface. The control registers are implemented on a Xilinx Virtex-II FPGA. The physical link between the host CPU and the Timing Module is a 66 MHz PCI-X bus. The PCI interface is implemented with the

Xilinx PCI-X IP core as a 32-bit endpoint device. The PCI-X interface, as implemented, is not able to satisfy the growing bandwidth requirements for the SNS systems, which results in the loss of timing data. It is practically impossible to achieve additional bandwidth and higher data rates with the existing PCI-X interface. Apart from the communication bus bandwidth limitation, there is also a logic area constraint in the Virtex-II FPGA making it impossible to add more functionality and registers to the current design. Originally, a single Timing Module was intended to control eight chopper systems, but the logic constraints have limited the module to only four chopper controls. To address the listed problems in the Timing Module, we consider redesign of the current module. In the redesign, we modify the PCB and FPGA design of the current Timing Module to allow operation on a PCI-Express bus thereby improving the bandwidth. Additionally, we implement the Timing Module logic in a larger FPGA that includes communication macros for the PCI Express interface and provides more room for additional functionality.

The optical communication card (OCC) [4, 5] is an interface to the high-performance computers for communication to the SNS detector electronics. The detector electronics timestamp the events and calculate the position index of the neutrons detected. Two 32-bit values determine the time event and position of a detected neutron. These values are stored locally in the detector memory and sent as a data packet 60 times a second. Up to 1 million bytes may be transferred during a frame. The OCC receives this data sent by the detector electronics via a fiber optic data link. The OCC also uses the Xilinx PCI-X 64bit/66MHz core (similar to that used in the Timing Module) for the communication with the host CPU. The OCC is capable of initiating a DMA transfer of the data from the detector electronics to the memory of the host computers. The DMA transfer capability of OCC indicates that it has higher data rate requirements than the Timing Module. Therefore, like the Timing Module design, we modify the current FPGA design to use the PCI Express interface and bus master DMA operation.

The FPGA-based PCI Express implementation uses PCI Express IP cores available from Xilinx. The advantages of an FPGA-based solution are that it allows the designer to create a design that exactly matches the user's requirements and provides flexibility in the design making an FPGA-based design an ideal solution for the SNS timing and communication cards. There are three options when an FPGA is used for PCI-based designs: one-chip solution with a soft IP core, one-chip solution with a hard IP core [6], and two-chip solution [7]. The one-chip solution with a soft IP core utilizes a high-performance FPGA such as Xilinx Virtex-4 to perform the PCI Express protocol, physical interface, and transmission and the soft IP core is implemented in the FPGA logic. The one-chip solution with a Hard IP core utilizes a Virtex-5 FPGA, which includes a PCI Express endpoint block in hard logic. Xilinx provides PCI Express endpoint solutions to configure the Virtex-5 Built-in Endpoint Block. These solutions require additional FPGA logic to create a complete Endpoint solution for the PCI Express operation. In the two-chip solution, the physical layer resides in a dedicated chip c (PHY) [7] and the logic and the transport layers reside in an FPGA. There is an interface between the external PHY device and the FPGA device called a PIPE (Physical Interface for PCI Express). A broad range of PHY devices are available from manufacturers such as Genesys Logic, Philips Semiconductor, and Texas Instruments. This two-chip solution uses a low-cost FPGA such as a Xilinx Spartan-3 connected over a PIPE interface to the PCI Express PHY chip to implement all layers of the interface. Advantages of the single-chip solution include higher performance and simplified PCB while the two-chip solution is more cost effective. In order to obtain better performance and more functionality, we selected Xilinx Virtex-5 FPGA with PCI Express hard IP core.

In our work, we primarily perform the following tasks to modify the current Timing Module and OCC:

1) Circuit design,
2) FPGA code design,
3) Debug and troubleshooting of the design to assure design goals have been met.

In the circuit design phase, we reviewed the PCI-X design and investigated the bus changes needed for PCI Express. After deciding on the Xilinx FPGA and the PCI-Express solution, we provided circuit board design assistance to the SNS DAS group. In the FPGA code design phase, we reviewed the current VHDL code and modified the code to work with the Xilinx Virtex-5 PCI Express hard IP core. Debugging and troubleshooting was conducted throughout the design of the FPGA code and continued through the debugging of the new board and system testing. The debugging was performed with test benches, simulations, and JTAG signal capture.

The thesis is organized in the following manner: Chapter 2 provides the existing design overview and requirements, Chapter 3 discusses the PCI Express architecture and the Xilinx programmed I/O and DMA example designs used as references to integrate the PCI-Express with the existing design. Chapter 4 discusses the design and implementation methods. Chapter 5 provides experimental and performance results. Chapter 6 concludes the thesis.

CHAPTER TWO

BACKGROUND

In this chapter, we discuss the existing PCI-X based Timing Module and the Optical Communication Card (OCC) designs that facilitate neutron data acquisition. We also discuss the theory of operation, block diagrams, and register information for the boards.

2.1 Timing Module Technical Details

At the SNS neutron scattering facility, the instruments use time-of-flight (TOF) measurements to calculate the energies of the detected neutrons. TOF is the time difference between neutron generation and the time at which the neutron is detected by the detector electronics. The Timing Module [2] [3] in the SNS facility is responsible for providing timing signals to the detector electronics. The Timing Module also provides signals to the chopper system [8] for phasing of the chopper disk with the proton on target event. A chopper in the instrument system is designed to slow down fast neutrons in a prompt pulse. The Timing Module can also help with analysis of the chopper performance. The Timing Module communicates with the detector electronics [9], the chopper timing system, and the accelerator timing system via fiber optic links. The Timing Module registers implemented in the FPGA are divided into three major subsystems: Tsync generation circuit, veto generation circuit, and phase and timing circuit. The Timing Module also handles the multi-frame timing requirement for multiple protons in the beam line. Finally, the Xilinx FPGA also implements the PCI-X soft core and interface registers.

Figure 2.1 shows the Timing Module printed circuit board (PCB) and Figure 2.2 shows the functional block diagram of the Timing Module. The diagrams show three optical transceivers and the Virtex-II FPGA for the timing logic and the PCI-X interface. The PCI-X interface uses the Xilinx PCI-X logic IP core [10].

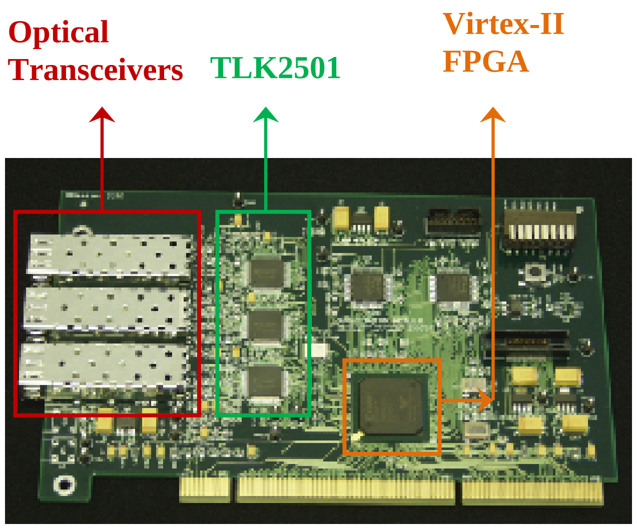


Figure 2.1: Timing Module PCB

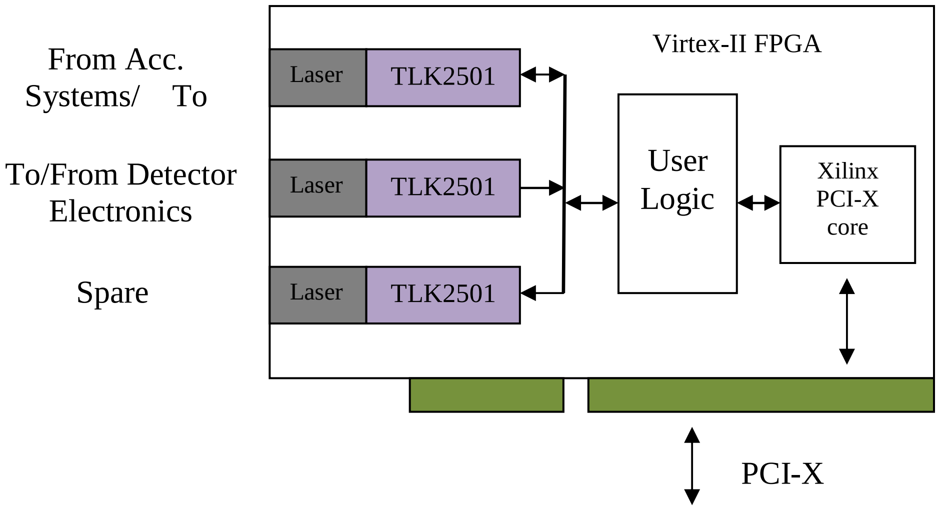


Figure 2.2: System blocks of Timing Module

2.1.1 Optical Link Signals

There are three optical transceivers in the Timing Module board. The first optical link is used to receive signals from the accelerator system and to send signals to the chopper system, forming a timing loop described in section 2.1.5. The second optical link is responsible for sending signals to the detector electronics. The third optical link is a spare link for testing.

The timing loop input signals are given below:

- PT0 or proton on target signal is an approximately 1 us pulse indicating the start of the proton beam on target. It periodically occurs every 16.67 ms.
- Tstart signal is an approximately 1 us long pulse that indicates the time of the extraction event.
- Beam veto signal is an approximately 1 us long pulse that indicates when the beam has been dumped and no proton pulse will be on the target.
- Loss of lock signal indicates that the master timing unit has lost lock.
- The chopper Top Dead Center (TDC) pulse is an approximately 1 us long pulse that indicates when the chopper has reached a position known as TDC. There may be up to 8 TDC pulses for the 8 chopper systems.

The timing loop output signals are given below:

- Chopper reference pulse is an approximately 200 us long pulse that is phased with the Tstart signal generated by the accelerator. This signal is used by the chopper control system to synchronize the phasing of the chopper disk.

- Tsync pulse is an approximately 1 us long pulse to the detector electronics. The signal indicates that the detector electronics should reset the time stamp counts.
- Veto pulse is an approximately 1 us long pulse to the detector electronics. It indicates that the detector electronics should veto the neutron data from the current neutron pulse.

2.1.2 FPGA User Logic

The user logic is implemented in a Xilinx Virtex-II FPGA. It generates the appropriate signals for the detector electronics in the current neutron frame such as the veto and Tsync signals. It is also responsible for maintaining the timing and phase information on the accelerator and chopper pulses. The Timing Module registers implemented in the FPGA are in little endian format. Figure 2.3 shows the input and output signals to the FPGA. The Timing Module receives PT0, Tstart, Beam Veto, and Loss of Lock input signals from the accelerator systems. As shown in Figure 2.3, the Timing Module also receives chopper TDC pulses from chopper systems and provides chopper reference pulses to the choppers. The Tsync and Veto output pulses are transmitted to the detector electronics.

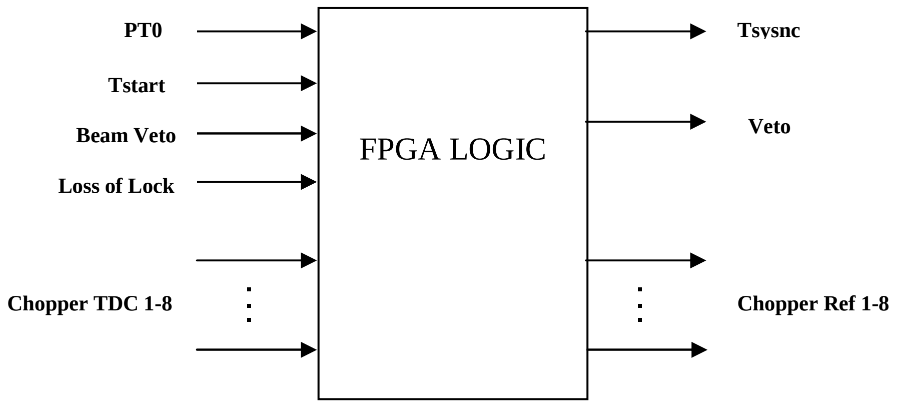


Figure 2.3: FPGA logic I/O signals

2.1.3 Multi-Frame Timing Requirement

In many SNS instruments, the lengths of the beam lines are long enough to hold more than one neutron in a single beam line. The Timing Module FPGA is responsible for handling these multi-frame situations. It also maintains the time difference between proton pulses to correctly calculate TOF.

Figure 2.4 shows the short instrument system operation. In this case, the neutrons are detected (Tdet) prior to the generation of the next neutron pulse (the grey pulses in Figure 2.4). The pulses on the time axis represent PT0. The slots at the chopper position indicate the position of the chopper opening. The TOF is determined by adding the time stamp Tdet at the detector electronics and the offset, which is a small correction due to the finite speed at which the PT0 pulse is received by the detector electronics.

Figure 2.5 shows the long instrument operation. In this scenario, a veto in frame n will be associated with the neutron detection occurring in frame n+1. In this case, calculation of the TOF requires the addition of the time PT0(n+1) – PT0(n) term to the time stamp generated at the detector electronics.

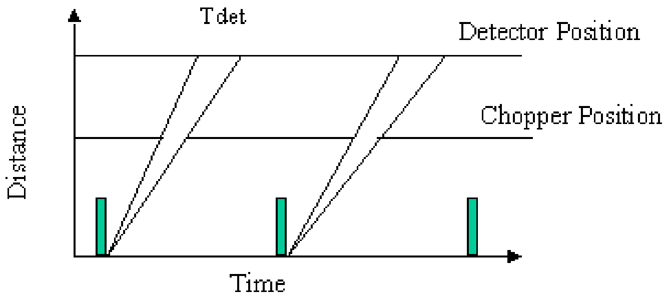


Figure 2.4: Short instrument system neutron detection

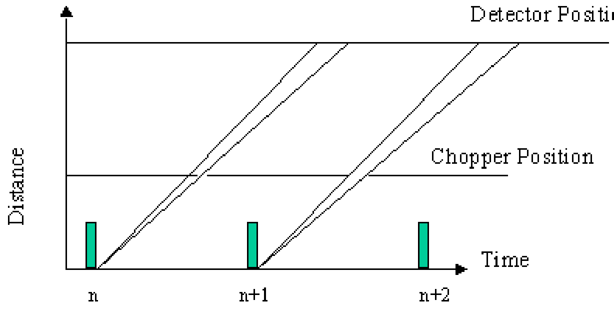


Figure 2.5: Long instrument system neutron detection

2.1.4 Timing Module Subsystems

The three subsystems of the Timing Module logic are timing and phase register circuitry, veto generation circuitry, and Tsync generation circuitry.

The timing and phase register circuitry, as shown in Figure 2.6, contains PT0 to PT0 time registers and chopper phase registers. The PT0 to PT0 time registers keep track of the successive PT0 times. There are 16 such registers stored in a block RAM memory. The first register keeps time difference between PT0 of the nth pulse and PT0 of the (n-1)th pulse. Similarly the second register keeps track of the time difference between PT0 of the nth pulse and PT0 of the (n-2)th pulse and so on. The chopper phase registers keep track of the time difference between successive TDC pulses of the corresponding choppers.

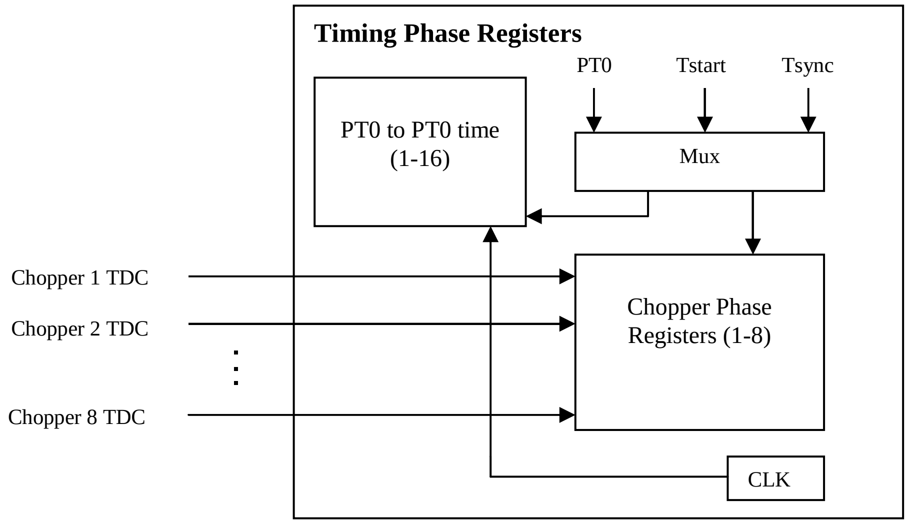


Figure 2.6: Timing and phase register circuitry

The veto generation circuitry, as shown in Figure 2.7, consists of the registers containing the beam veto mask, PT0 veto mask, and chopper veto masks. The beam veto mask register generates veto pulses when the beam veto signal occurs. The PT0 veto mask register controls the generation of the veto pulses whenever the PT0 overdue counter from the Tsync generation circuit triggers. The chopper veto mask registers are responsible for generating the veto pulses for the detector electronics when the chopper veto is seen.

The Tsync generation circuitry, as shown in Figure 2.8, consists of the Tsync delay registers, PT0 overdue time, free running divisor, and Tsync control registers. The Tsync delay register will delay the output by an amount up to 16.7 ms in intervals of 9.42 ns steps. The PT0 overdue time is loaded into a PT0 overdue counter whenever a PT0 or

Tstart is seen. The free running divisor generates a 60 Hz clock that can be used in place of Tstart and PT0 for testing purpose.

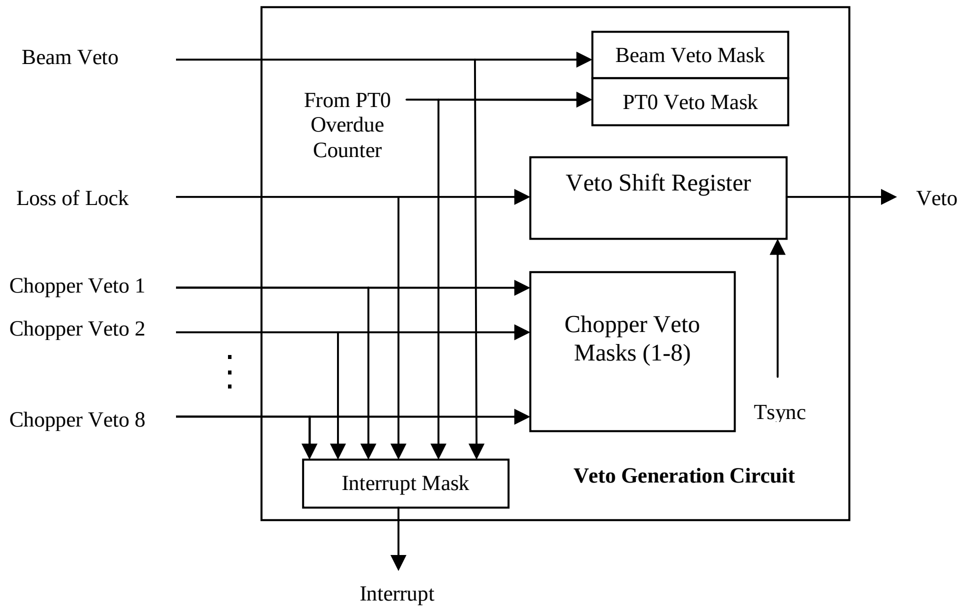


Figure 2.7: Veto generation circuitry

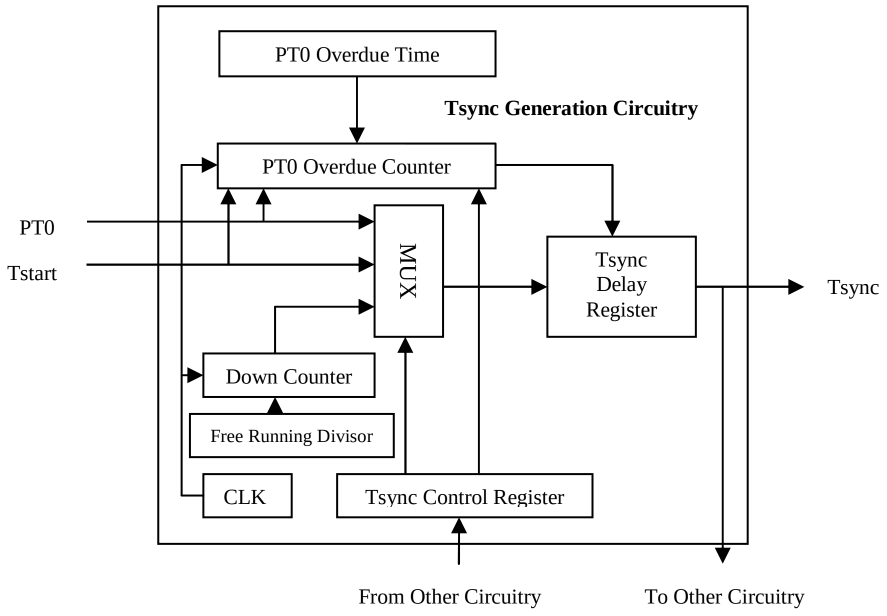


Figure 2.8: Tsync generation circuitry

2.1.5 Timing Loop

The timing loop, as shown in Figure 2.9, is a closed loop created between the chopper Optical Distribution Module (ODM), accelerator ODM, and the Timing Module. The ODM is an optical transceiver that converts digital signals to optical pulses. It also provides the inverse function of restoring optical pulses to digital outputs. The chopper TDC pulses from the chopper ODM are passed though the optical link to the accelerator ODM. The Tstart and PT0 signals from the accelerator are fed into the accelerator ODM and sent to the Timing Module with the chopper TDC pulses. All the above signals are used by the Timing Module to produce vetoes and pulse-by-pulse chopper phase error and reference signals that are passed both to the detector ODM and back to the chopper ODM.

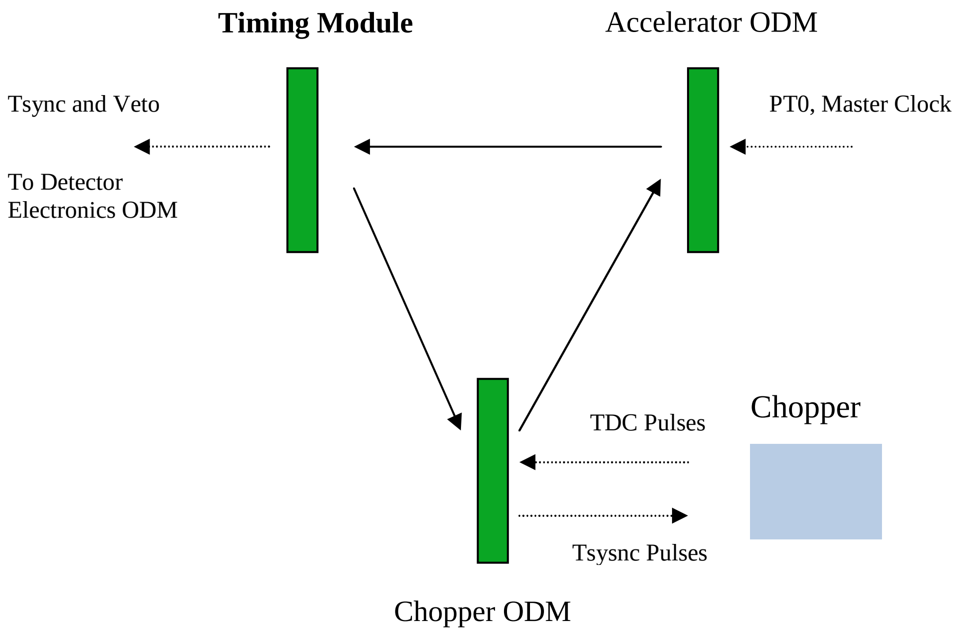


Figure 2.9: Timing loop

2.1.6 PCI-X Register Interface

All registers in the Timing Module are defined as 32-bits. However, not all registers use the full 32 bits assigned to them. The number of registers could be reduced by packing those registers that are shorter than 32-bits. This packing option is necessary in the Xilinx Virtex-II FPGA due to the logic constraints. However, register packing results in a more complex and less intuitive design. In the new Xilinx Virtex-5 design, there is ample logic and all registers are assigned the full 32 bits.

The registers are divided into two categories: Read/Write registers and Read-only registers. The Read/Write registers are used for configuration and control. These registers are accessed by the Windows-based software application through a device driver to configure and control the operation of the Timing Module. The Read-only registers provide the timing information, error information, and status of the Timing Module. Some of this information is required by the Data Acquisition Systems (DAS) operation while other information is required when debugging the Timing Module.

2.2 Optical Communication Card Technical Details

The Optical Communication Card (OCC) [4, 5] receives data from the detector electronics via a fiber optic data link using a lightweight point-to-point protocol. There are two 32-bit registers kept locally in the detector electronics that store the time stamp and position index of the detected neutrons. The register values are sent to the OCC periodically at a rate of approximately 60 times a second. Figure 2.10 shows the block diagram of the link between detector electronics and the OCC.

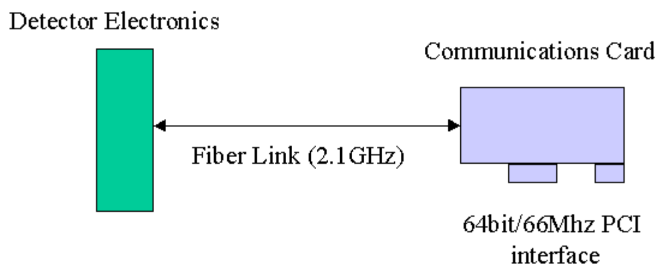


Figure 2.10: OCC and detector electronics connection

2.2.1 Optical Communication Card Block Architecture

Figure 2.11 shows the OCC printed circuit board with all the hardware components and Figure 2.12 shows the block diagram of the optical communication card.

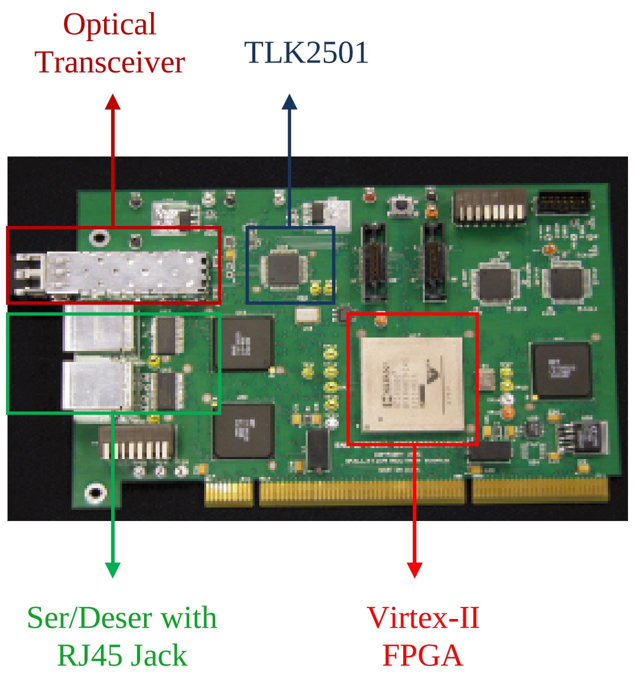


Figure 2.11: OCC PCB

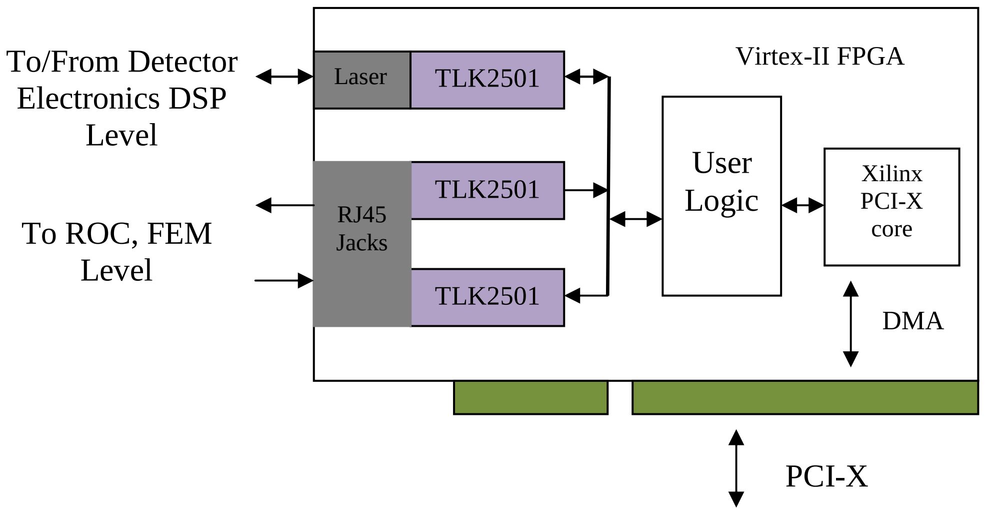


Figure 2.12: System blocks of OCC

The interface logic consists of the Xilinx PCI-X 64bit/66MHz soft core on a Xilinx Virtex-II FPGA. The control logic consists of the control and status registers and state machines that implement the communication protocol for the optical and copper transceiver chips. The control logic is also responsible for reading from and writing to the internal FIFOs and handling the DMA transfers.

The high-speed data links consist of 21 signals that are serialized into 3 LVDS signal pairs for transmission. The fourth LVDS pair transmits the data clock. At the receiving end, the 3 LVDS signals are de-serialized and recovered into the 21 data link signals using the data clock on the fourth LVDS signal. The high-speed data links are connected to the ROC [11], FEM [12], and DSP [13] boards, which are interfaced to the detector electronics. The optical interface is handled by an optical transceiver through a TLK2501 chip [14].

2.2.2 Communication Protocol

This section describes the communication protocol related to the PCI bus, the optical interface, and the LVDS interface. Data width for the PCI bus is 32-bits wide. The data sent via the optical interface and the LVDS interface is not formatted. The inbound data from the LVDS link has minimal processing, while the data from optical communication link is processed according to the SNS protocol format discussed in the following subsections.

2.2.2.1 OCC Memory

The control and status registers are referenced to BAR0 of the OCC memory space. The other memory areas IDMA or input FIFO, ODMA, and output FIFO are implemented on the FPGA as a circular buffer with producer and consumer index and referenced to BAR1 of the OCC memory space. These memory areas store data that are transferred between the LVDS or optical communication interface and the host PC.

For target write, the OCC IDMA memory area is written. The memory area is a 32-bit addressable dual port memory with 8 KB length. For target read, the OCC ODMA memory is written. The memory is a 32-bit addressable dual port memory with 16 KB length. The memories are implemented as a circular buffer to allow reading and writing of the ODMA and the IDMA memory areas. The circular buffers use the producer and consumer index. The producer index keeps track of the writing position and the consumer index keeps track of the reading position of the buffer. The buffer is considered empty when both the indexes are equal and full when the producer index is one less than the consumer index. The output FIFO consists of the Inbound Message Queue (IMQ), Data Queue (DQ), and Command Queue (CQ).

2.2.2.2 LVDS Interface Protocol

Figure 2.13 shows the data communication between the LVDS interface and the host PC. During a target write, the data is transferred from the host PC to the OCC and then sent over the LVDS link. The data is first written to the IDMA at the current producer index. Then the producer index and the IDMA length are updated. The software application writes to bit 0 of the configuration register which sets the TX-GO bit in the firmware. Then the OCC firmware sets TX_IP bit automatically, which indicates the that the transmission is in progress. When the transmission cycle is done, a hardware interrupt occurs.

During a target read, bit 1 of the status A register is set to indicate that the received data is available from the LVDS link. The data is transferred from the OCC without a DMA transfer. The target ODMA length register contains the number of bytes to read. If desired, an interrupt can be enabled whenever data is available for reading. In the case of initiator writes, the data at the LVDS deserializer is connected to the output FIFO. The data transfer from the output FIFO to the host PC memory is discussed in the section 2.2.2.3. The FPGA keeps track of the input count. Later the data from the output FIFO is transferred to the system memory through DMA transfer.

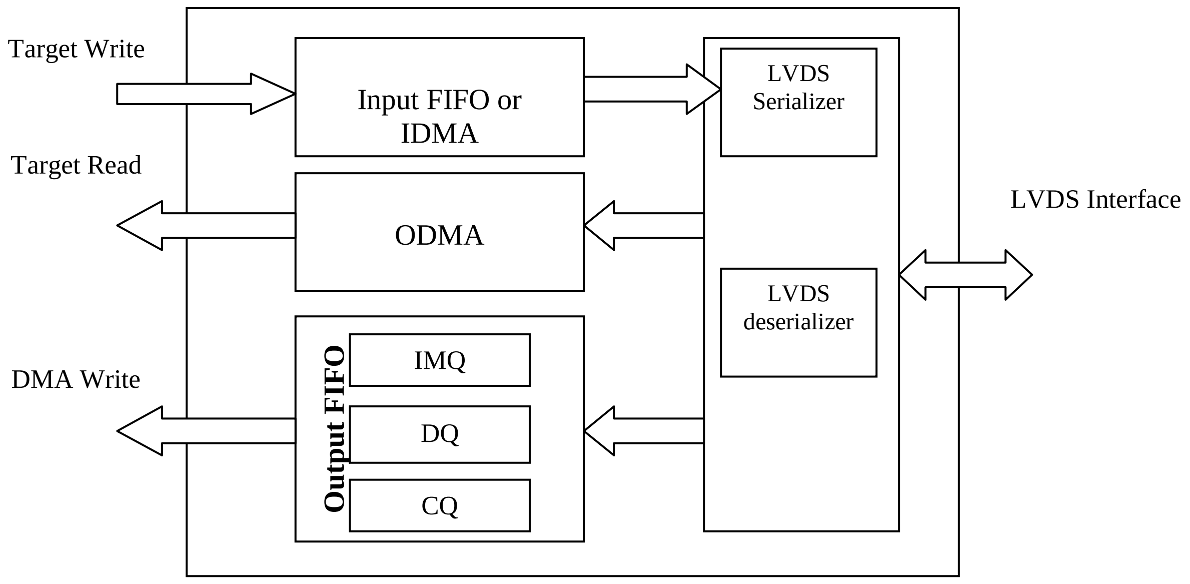


Figure 2.13: Data communication between LVDS interface and host PC

2.2.2.3 Optical Interface Protocol

For the transmission of the data between the OCC and the optical communication link, the OPTCVR bit in the control register is set. When data is written from the host PC to the OCC card via the PCI bus, it is written to an internal input FIFO on the OCC. When TX_GO bit is set through the control register, the data transfer from input FIFO is initiated over the optical communication link. An interrupt can be generated after the successful completion of the entire DMA transfer. Figure 2.14 shows the data transfer between the optical communication interface and the host PC.

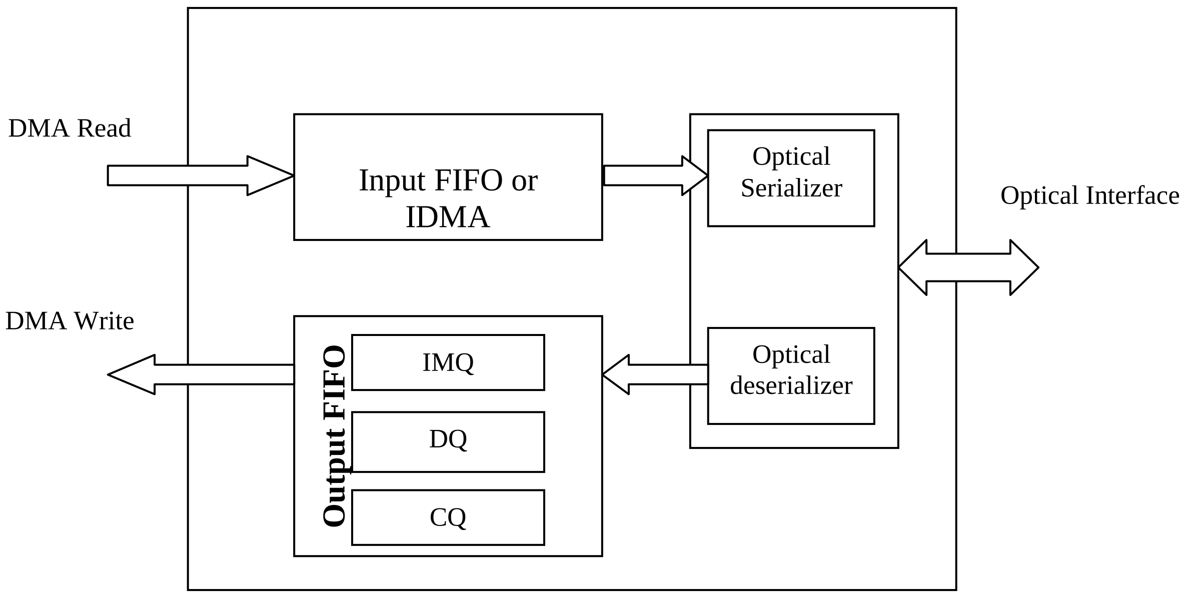


Figure 2.14: Data communication between optical interface and host PC

The optical serializer is a 16-bit data interface used to transfer 32-bit data. When the data is sent from the input FIFO to the optical communication card, the lower 16-bits are sent in the first clock cycle followed by the upper 16 bits in the next clock cycle. Data from the optical deserializer is connected to the output FIFO implemented on the FPGA, as shown in Figure 2.14. The FPGA keeps track of the input count and also keeps track of the reception of the command descriptor which decides whether the packet received is a command packet or a data packet.

The data transfer from the output FIFO to the host PC memory via the PCI bus is done in one of the following two ways:

- The OCC receives the entire data packet including the header and payload DWORDs. The header is written to the Inbound Message Queue (IMQ) of the host PC via the PCI initiator protocol. The payload is sent to the Data Queue (DQ). The DQ is a circular buffer of length N where N is power of 2. The maximum limit of the DQ in length is 4 GB. The DQ producer index is a circular offset pointer to this memory and is incremented on quadword boundaries. The incremented DQ producer index and the IMQ producer index are written to the host at the DQ producer address and the IMQ producer address respectively. An interrupt is set to inform the host to process the data at DQ.

- When the OCC receives a command packet, the header is written to the IMQ of the host via the PCI bus. The payload is written to the Command Queue (CQ). The CQ is also a circular queue but much smaller that the DQ. The incremented CQ producer index and the IMQ producer index are written to the host at the CQ producer address and the IMQ producer address respectively. An interrupt is set to inform the host to process the data at CQ.

2.4 Summary

This chapter discussed the original PCI-X based Timing Module and OCC design. We thoroughly studied the earlier designs to migrate the design to work with the Virtex-5 FPGA and PCI Express. We discussed all the registers and memory buffers required for the Timing Module and OCC implementation. Our approach to integrate the PCI Express bus in the designs was also discussed. Chapter 3 discusses the PCI Express architecture and Xilinx Endpoint Block Plus for PCI Express used in the new design.

CHAPTER THREE

PCI EXPRESS ARCHITECTURE

This chapter discusses the PCI Express architecture and Xilinx Endpoint Block Plus solution for PCI Express. It also discusses the programmed input/output (PIO) [19] and bus master DMA (BMD) example design [20] available from Xilinx.

3.1 PCI Express Architecture

The PCI Express architecture includes the PCI Express topology, the PCI Express Endpoint that is implemented as three-layer architecture, and the software compatibility.

3.1.1 PCI Express Topology

PCI Express is a point-to-point serial interconnect that provides high-bandwidth communication over fewer pins than older PCI implementations. The PCI Express topology [15], shown in Figure 3.1, is composed of a root complex, several endpoints (I/O devices), and a switch.

PCI Express connects the CPU and memory subsystems to the I/O endpoints through a switch. An endpoint refers to a device that acts as a requester and/or completer to the PCI Express transaction. An Endpoint is an I/O device connected to the PCI Express, such as the Timing Module and the Optical Communication Card (OCC) in our case. Each port of the switch is connected to a PCI Express endpoint or a legacy endpoint. A switch enables a series of connectors for add-in I/O and appears to the configuration software as two or more logical PCI-PCI bridges. The switch uses address base routing to forward transactions.

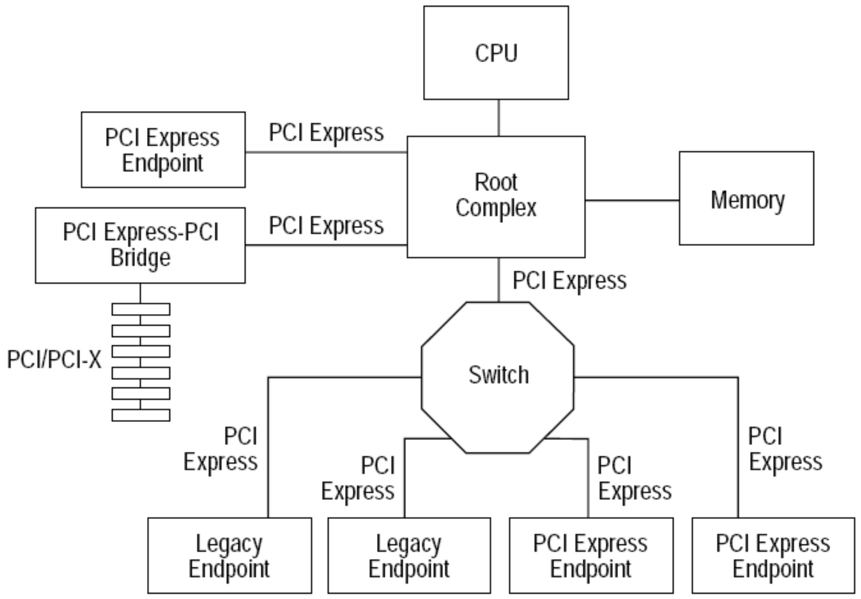


Figure 3.1: PCI Express topology

3.1.2 PCI Express Layering

The PCI Express protocol has three discrete logical layers: the transaction layer, the data link layer, and the physical layer. Each layer can be divided into two sections for data flow as shown in Figure 3.2 [15]: the transmit section and the receive section.

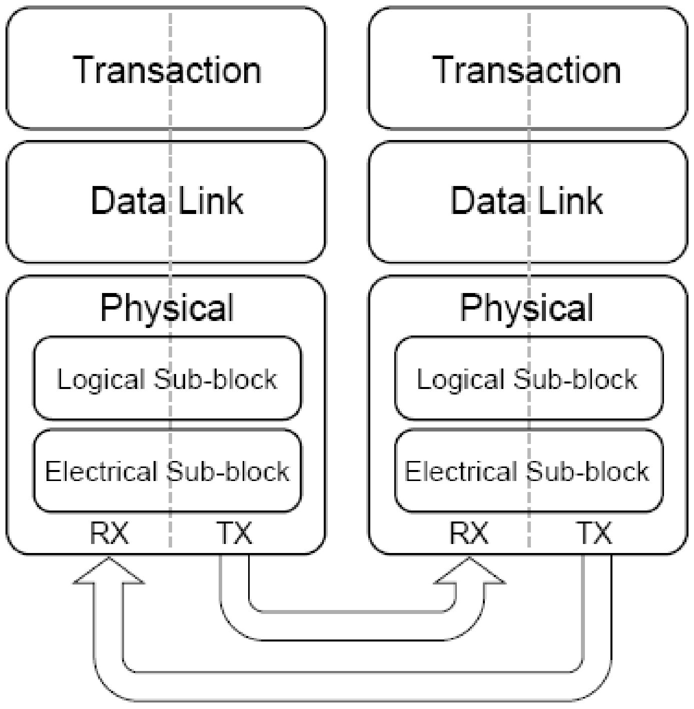


Figure 3.2: PCI Express data flow

PCI Express uses packets to communicate between layers. The major function of the protocol layers is to generate and process Transaction Layer Packets (TLPs). Figure 3.3 [15] shows the TLP packet flow through the layers. Other functions of the protocol layers include flow control management, initializing and power management functions, data protection, error checking and retry functions, and maintenance and status tracking.

The transaction layer is the upper layer of the architecture and its primary function is to assemble and disassemble the TLPs. Read, write and certain types of PCI Express events happen through TLPs. The transaction layer supports three PCI address spaces: memory, I/O, and configuration.

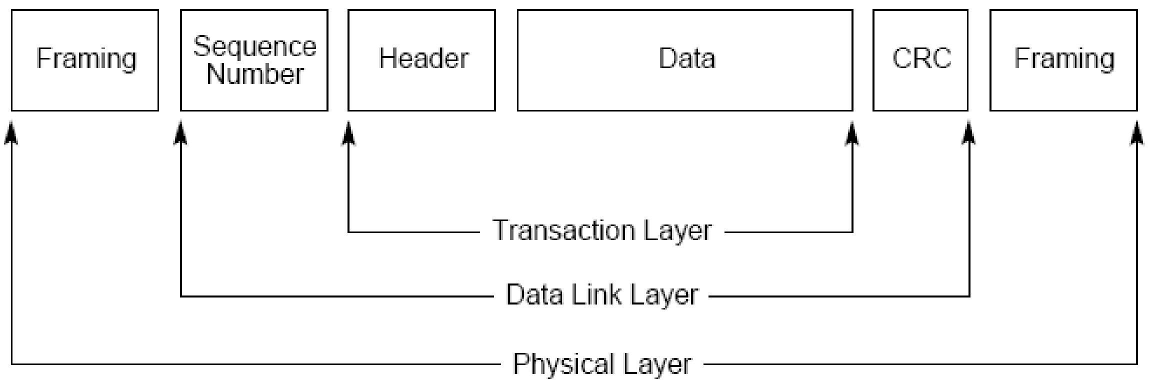


Figure 3.3: TLP flow through the layers

The Data Link layer is the intermediate layer between the transaction and physical layers. The responsibilities of the Data Link layer are: link management, error detection, and error correction. The transmitting side of the Data Link layer accepts TLP from transaction layer. It applies the data protection code and TLP sequence number and passes it to the physical layer for physical transmission through the link. The receiving side of the Data Link layer checks the integrity of the received TLP and submits them to the transaction layer for further processing. If a TLP error is detected, the layer requests retransmission of the TLP until the TLP is received correctly, or the link is considered to have failed. The Data Link layer also generates and consumes its own packets called the Data Link Layer Packets (DLLP) that are used for link management functions.

The physical layer performs all interfacing operations, including driver and input buffers, framing, de-framing, parallel-to-serial and serial-to-parallel conversion, Phased Locked Loop (PLL), and 8b/10b encoding and decoding of TLPs and DLLPs. The physical layer also supports lane reversal for multi-lane designs and lane priority inversion.

3.1.3 Software Compatibility

PCI Express uses PCI compatible configuration and device driver interfaces [16] allowing backward compatibility with legacy PCI designs. The PCI architecture layer with PCI driver compatibility is shown in Figure 3.4. The PCI configuration space and programmability of I/O devices are unchanged in PCI Express. Therefore, current operating systems, which are PCI compatible can boot for PCI Express without any changes in the device driver. The run-time software model such as load-store and shared memory model of PCI is also maintained within the PCI Express architecture. New software may be developed to utilize new capabilities of the PCI Express architecture.

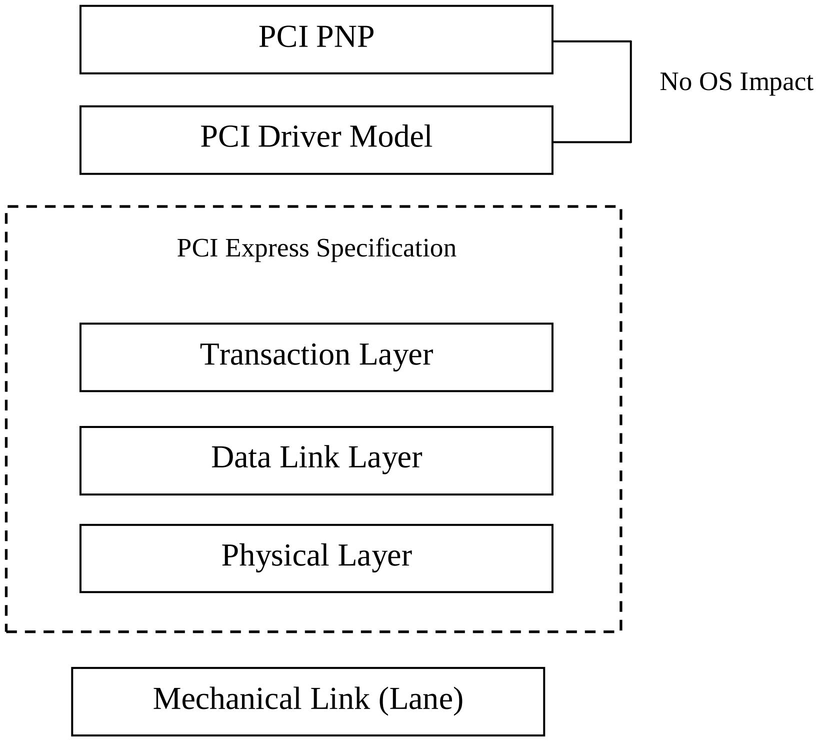


Figure 3.4: PCI Express architecture layer

3.2 Xilinx Endpoint Block Plus for PCI Express

The Endpoint Block Plus for PCI Express [17] from Xilinx is a scalable serial interconnect building block used with the Xilinx Virtex-5 series FPGAs. The solution supports x1, x4, and x8 lane options for the PCI Express base specifications v1.1. Table 3.1 shows the device support, resource usage, and data path width for the Endpoint Block Plus solution.

Figure 3.5 shows the top-level functional blocks and interfaces for the PCI Express Endpoint Block Plus core. The Endpoint Block configuration management layer implements the PCI Type 0 Endpoint configuration space providing the following functions:

- PCI Configuration Space
- Power management
- Error reporting and status functionality
- Configuration Reads and Writes for receive
- Completion with or without data for transmit
- Interrupt emulation

Product	I/O	Device Support	LUTs	FFs	Data Path Width
1-lane Endpoint Block Plus	1	Virtex 5 LXT/SXT	2100	2250	64
4-lane Endpoint Block Plus	4		2100	2250	64
8-lane Endpoint Block Plus	8		2100	2250	64

Table 3.1: PCI Express lane overview

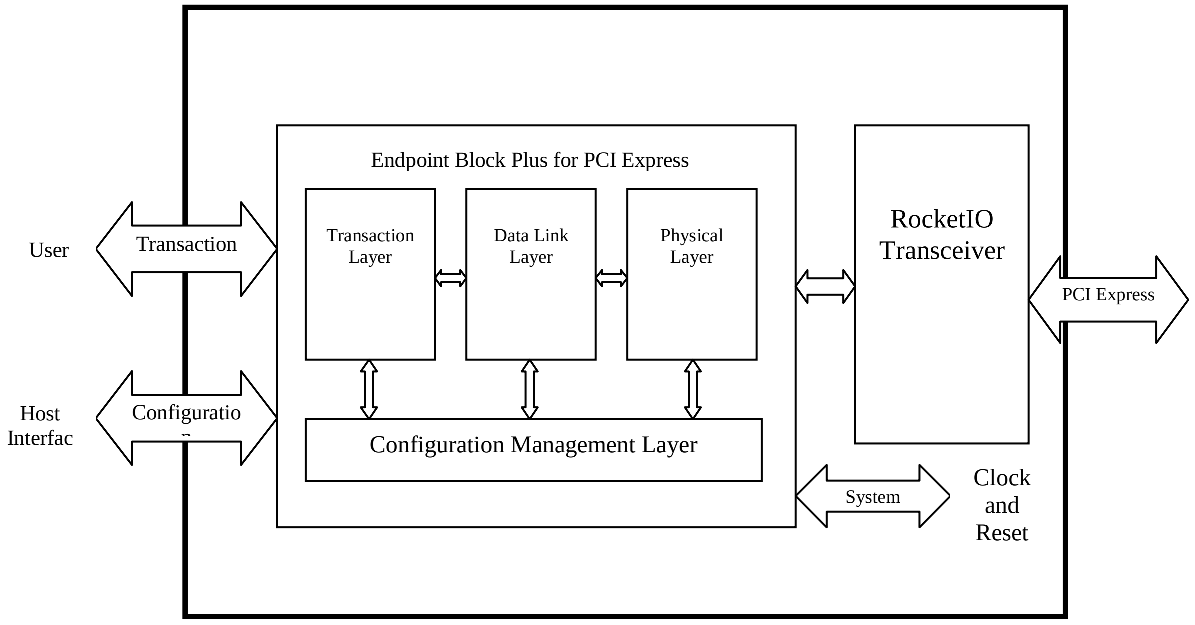


Figure 3.5: Top-level functional blocks of PCI Express endpoint

The Xilinx PCI Express Endpoint Block uses the RocketIO GTP transceivers [18] for the packet transaction. The RocketIO is a power-efficient high speed serial I/O available in the Virtex 5 FPGAs. Using these I/O pins, transceiver module is designed to operate at a serial bit rate of 2.5 GB/s for the PCI Express protocol.

The interfaces to the Block Plus core as shown in Figure 3.5 are defined below:

- System Interface
- PCI Express Interface
- Configuration Interface
- Transaction Interface

The system interface consists of the reset signal, clock signal, and a free running reference clock output signal. The assertion of the asynchronous system reset signal performs a hard reset of the entire core. The system clock signal is either 100 MHz or

250 MHz depending on the user’s selection. The PCI Express endpoint uses a synchronous clocked system as shown in Figure 3.6 where a 100 MHz clock from the link is fed to a jitter attenuator to get a 100 MHz or a 250 MHz input clock to the core.

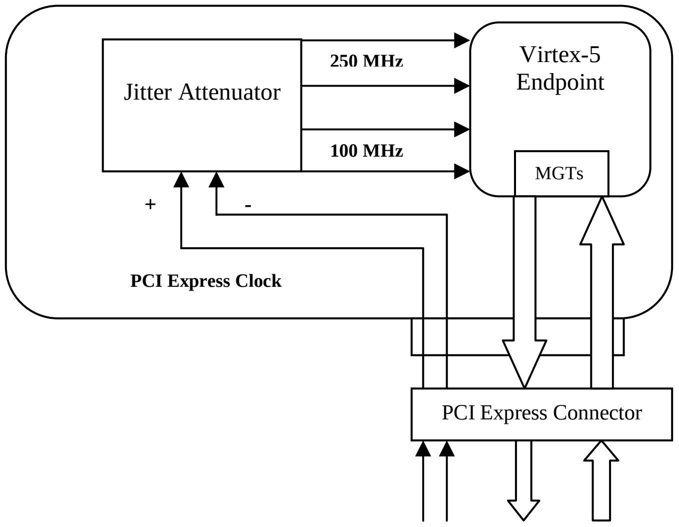


Figure 3.6: PCI Express clock

The PCI Express interface consists of a pair of transmit and receive differential signals. For example, the x1 core uses only lane 0 with a pair of differential signals for transmit and a second pair for receive. To achieve higher bandwidth, the x4 core uses lanes 0-3 and x8 core supports lanes 0-7.

The transaction interface is the interface to the user logic that allows the user design to generate and consume TLPs. The most common transaction interface signals are transaction clock, transaction reset, and transaction link-up. Transaction of the TLP is synchronous with the transaction clock. The transaction clock frequency can be fixed when generating the PCI Express core using Xilinx CORE generator. Recommended and optional frequencies for each lane width are shown in Table 3.2.

Lane Width	**Recommended Frequency (MHz)**	**Optional Frequency (MHz)**
x1	**62.5**	**125**
x4	**125**	**250**
x8	**250**	**250**

Table 3.2: PCI Express lane frequency

The transaction link-up signal is asserted when a connection is established between the core and the link partner. It is de-asserted when the link is lost due to an error on the transmission channel. Figure 3.7 shows a timing diagram where an approximate delay between the transaction reset (trn_reset_n) and transaction link-up (trn_lnk_up_n) signals is shown. There are several transmit and receive transaction interface signals that are responsible for sending and receiving a TLP from the user logic. These signals are discussed in detail in Chapter 5 of the thesis.

The configuration interface allows the user logic in the FPGA to access and examine the configuration space of the Endpoint for PCI Express. It also provides the link and device status through the configuration signal. The user logic can also initiate a message signaling interrupt (MSI) or legacy interrupts through the configuration interface. Interrupt operation and implementation is discussed further in Chapter 4 and 5 of the thesis.

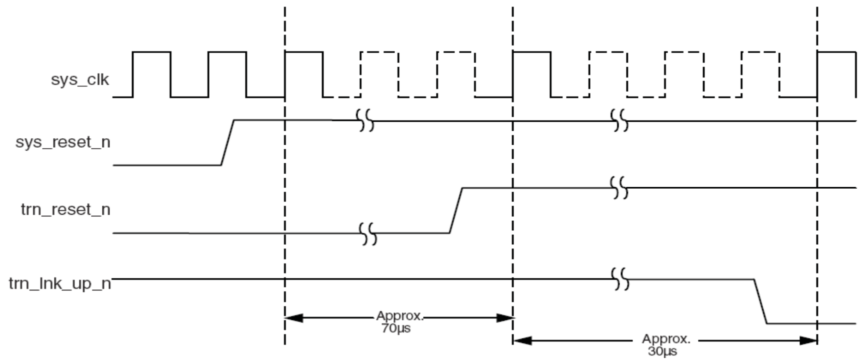


Figure 3.7: Transaction interface signal timing

3.3 Programmed I/O Design

A Programmed I/O (PIO) example design [17, 19] is used in this project to test and verify the basic PCI Express functionality in both of the new designs. The example design was also referenced closely when designing the Timing Module. The example design allows the host CPU to access the memory mapped input output (MMIO) and configuration mapped input output (CMIO).

The design consumes 8192 bytes of target space in the FPGA Block RAM. It supports one DWORD payload size with 32-bit or 64-bit addressing. In a typical write operation, the CPU issues a store register to a MMIO address command and the data is moved downstream. Then the Root Complex (see Figure 3.1) generates a Memory Write TLP with the specified MMIO location address, byte enables, and data payload. Finally, the Endpoint receives the Memory Write TLP and updates the corresponding local register to communicate that the transaction has been completed successfully.

In a typical read operation, the CPU issues a load register from a MMIO address command and the data is moved upstream. The Root Complex generates a Memory Read

TLP with the specified MMIO location address and byte enables. The Endpoint generates a Completion with Data TLP after receiving the Memory Read TLP.

3.3.1 TLP Flow in PIO

The PIO design processes a single DWORD payload in 32-bit/64-bit memory read/write and I/O read/write TLPs. The example design supports one I/O Base Address Register (BAR), one 32-bit Memory space, and one 64-bit Memory space.

In the case of a memory or I/O write, when the endpoint receives a TLP, the TLP transaction type is compared with the value in the core BAR. If the two values match, the TLP is sent to the receive module of the PIO design. The receive transaction interface has several handshaking signals that indicate the start of the packet, end of the packet, and packet ready. Along with the handshaking signals, the interface asserts the appropriate receive BAR hit signal (trn_rbar_hit_n[6:0]) to indicate the destination BAR for writing. The receive engine of the PIO design extracts the data and address fields and passes them to the internal block RAM write request controller.

In the case of a memory or I/O read, the TLP transaction type and address is again matched with the core BAR. If the TLP passes the check, the TLP is passed to the receive transaction interface of the PIO design. When the core asserts the specific receive BAR hit signal (trn_rbar_hit_n[6:0]), the receive engine determines the appropriate 2 KB block RAM to use. The receive state machine then collects the lower address bit from the memory/IO Read TLP and instructs the internal memory read request controller to start a read operation. Figure 3.8 shows the top-level functional blocks of the PIO design. The Timing Module design also implements a similar functional block for integrating the PCI

Express interface with the timing logic. This implementation is discussed further in Chapter 4.

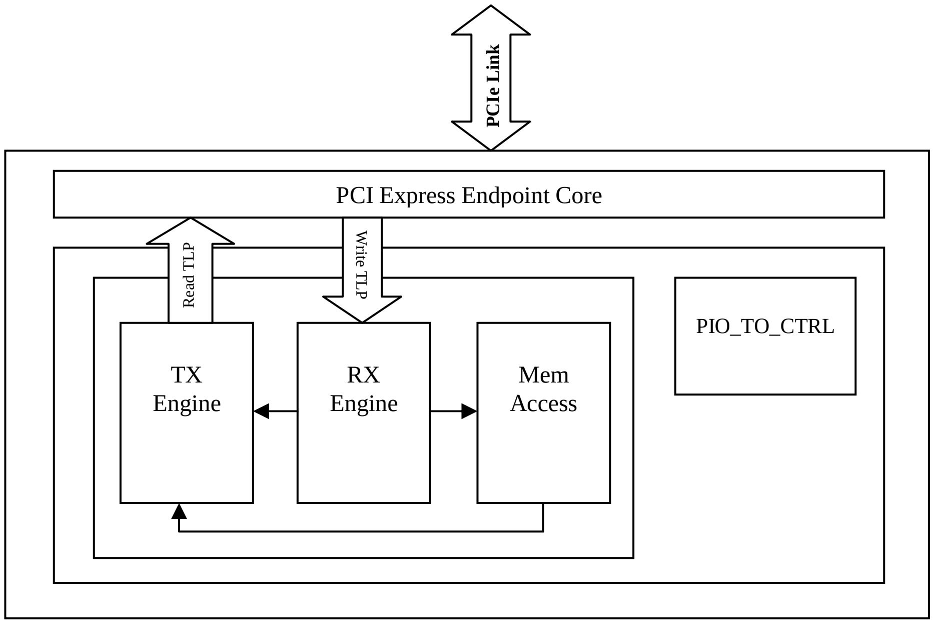


Figure 3.8: Functional block diagram of the PIO design

3.4 Bus Master DMA Design

The bus master DMA (BMD) design [20] moves data to and from host memory without the use of the CPU and is implemented with the Xilinx endpoint block plus core for PCI express. The core contains a DMA engine that controls memory writes and memory reads from the system memory. A requestor-completer model is used to transfer data from main memory to the endpoint. Memory write TLPs are used to transfer data from an endpoint buffer into main memory through the memory controller. Memory read TLPs are sent from the main memory to the endpoint. The TLP contains the starting address of the memory from which the DMA read will occur.

The bus master DMA design contains target logic, status/control registers, initiator logic, and interface logic along with endpoint block plus for PCI Express as shown in Figure 3.9 [20]. The target logic captures one DWORD memory write and memory read TLPs. It sends a completion with data signal in the case of the incoming memory reads. It also updates the status/control registers during writes and reads. The status/control register contains operational information regarding the DMA controller. The status registers are also used to measure the performance of the DMA transfer over the PCI Express bus. The initiator logic generates the memory write and memory read TLPs for transfer. A memory write TLP is generated when transferring data from the endpoint to the system main memory. The write DMA controller determines the address, size, payload content, and number of TLPs to be sent. An interrupt is generated when all of the memory TLPs are sent. Memory read TLPs are generated when transferring data from the system memory to the endpoint. The read DMA registers determine the address, size, payload content, and number of TLPs to be sent.

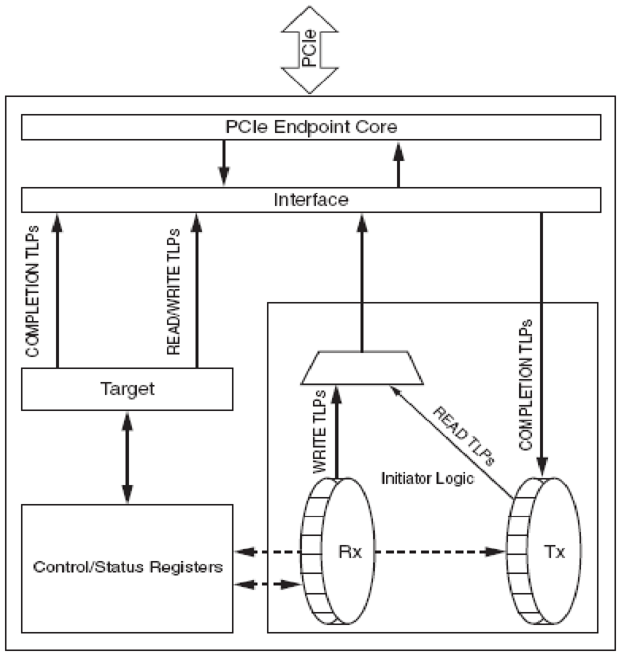


Figure 3.9: Bus master DMA architecture

3.5 Summary

This chapter presented the PCI Express architecture and key concepts for designing a PCI Express based system. We also reviewed the PCI Express base specification from PCI Special Interest Group (SIG) for details regarding transaction layer packets. Transaction layer specifications were examined for the interface design that connects the user logic with the Xilinx PCI Express endpoint core on the Virtex-5 FPGA. We also discussed the basic operation of the Xilinx PIO and BMD example designs. Chapter 4 discusses the design and implementation methods in detail.

CHAPTER FOUR

DESIGN AND IMPLEMENTATION

The operation speed of the two timing cards are limited by the current PCI-X bus implementation. To achieve the desired operation speed, the designs should be migrated to the high-bandwidth PCI Express bus. Our initial work involved the selection of a suitable FPGA-based PCI Express solution for the SNS timing cards. We considered the Endpoint Block plus Wrapper for PCI Express Xilinx IP core provided in the Virtex-5. The FPGA built-in Endpoint Block for PCI Express includes additional logic to create a complete solution for the PCI Express interface. We used an Avnet PCI Express development kit to test an example design for simple read and write operations. The board also confirmed the compatibility of the SNS PCI-X software application with the PCI Express operation. The board schematic also aided in designing the layout for the new Printed Circuit Board (PCB). The new board design also avoided the use of two separate custom cards for both Timing Module card and OCC. A daughter card slot was also designed in the new board for replacing the accelerator cards used to read accelerator links. Verification and debugging of the hardware was performed prior to the final layout of the board. On recommendation of the circuit changes for the PCI Express operation with the current design, the modifications were made to the existing VHDL code.

The implementation is divided in two parts, as shown in Figure 4.1: software and hardware. The hardware part consists of a PCB with a Xilinx Virtex-5 FPGA. The software part consists of the SNS PCI Express driver and the SNS custom application to initiate read/write commands.

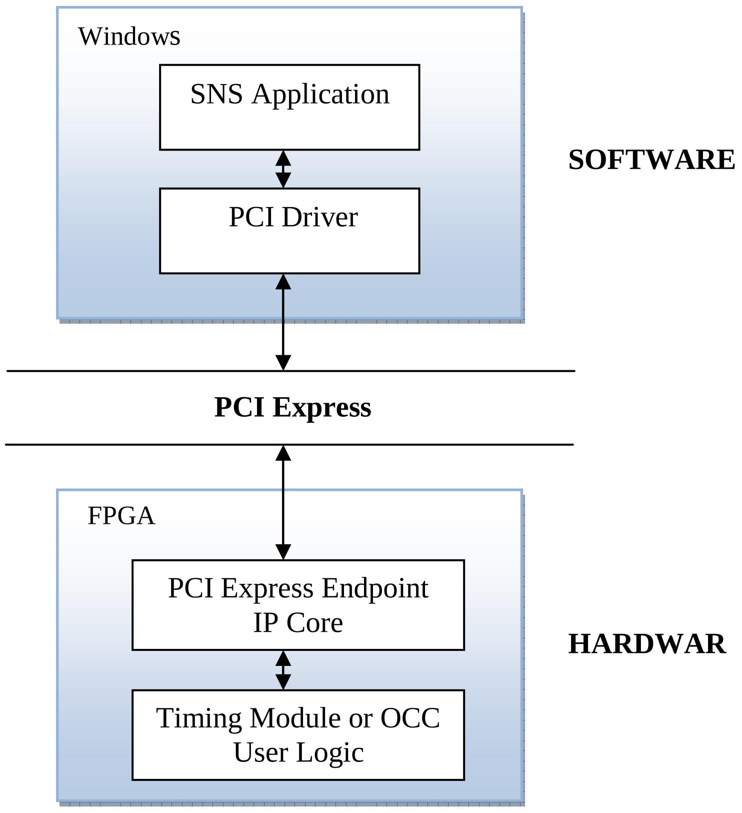


Figure 4.1: The entire system diagram

4.1 Hardware Systems

The hardware system consists of the new printed circuit board and FPGA firmware designed for both the Timing Module and OCC. The custom board is designed at ORNL with collaboration from Clemson University. The board consists of a Xilinx Virtex-5 FPGA that implements the Timing Module logic and OCC logic, the PCI Express interface, and the PCI express core. The other major components on the board such as the optical link, the transceivers chips and the LVDS links are for communication with the SNS instruments. The PCB will also add a daughter card in the future to replace an Accelerator Timing Card (ATC). Figure 4.2 shows the new fabricated Timing Module board that also implements the OCC design. In this section, we also discuss the FPGA firmware implementation for the PCI Express operation.

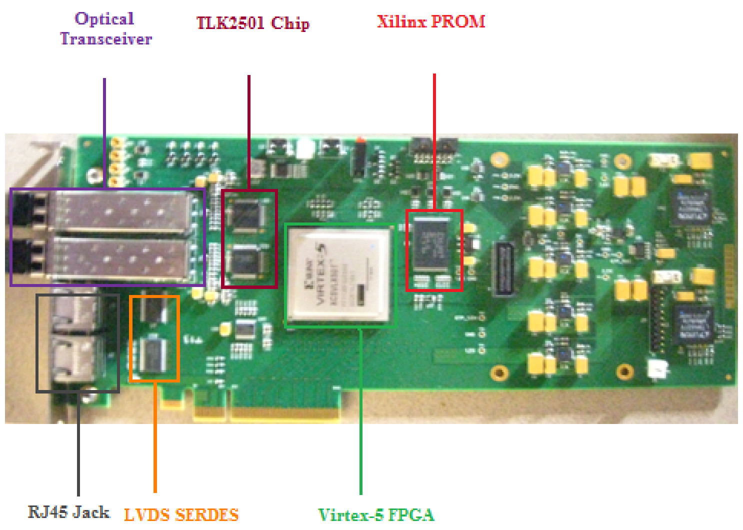


Figure 4.2: Timing Module printed circuit board

4.1.1 FPGA Device

The Xilinx Virtex-5 FPGA [21] was selected for the new SNS timing cards due to higher logic capacity and built-in PCI Express compatibility. Table 4.1 shows the XC5VLX50T FPGA device and resources available on the device.

Device	Configurable Logic Blocks (CLBs)		Block RAM Blocks			PCI Express Endpoint Block	Maximum RocketIO GTP Transceivers	Max User I/O
	Virtex-5 Slices	Max Distribution RAM (KB)	18 KB	36 KB	Max (KB)			
XC5VLX50T	7200	480	120	60	2160	1	12	480

Table 4.1: Virtex-5 XC5VLX50T device resource table

The Virtex-5 FPGA [21] has the following advantages over the Virtex-4 FPGA [22] that was also considered:

- The Virtex-5 device has a built-in PCI Express endpoint block that does not require the procurement of a separate soft IP core.

- The Virtex-5 FPGAs have 6-input look up tables (LUTs) compared to 4-input LUTs in earlier FPGAs. Therefore it provides more logic resources and is capable of higher complexity designs desirable for the future design expansion of the user application.

- The Virtex-5 uses a Phased Lock Loop (PLL) technology for lower jitter clock generation, which is very important for timing signals.

- The Virtex-5 has a 1.0V VCCINT instead of Virtex-4's 1.2V, which consumes less power.

4.1.2 Daughter Card

The new PCB design provides expansion capabilities through a customized daughter interface header card on the board. The daughter card will replace a rack-mounted PC and Accelerator Timing Card (ATC) that are currently used to read two accelerator links: the Real Time Data Link (RTDL) and Event Link (EvLNK). The EvLNK is a data link signal that triggers when a selected event is decoded on the data stream. The 60 Hz Tsync signal used by the Timing Module is received through this EvLNK data link. The RTDL collects the pulse ID, the type of pulse, and the number of stored turn's from the accelerator and forwards them to the DAS software applications.

Replacing the ATC and PC with a daughter card eliminates a lot of unnecessary hardware. Since the daughter card will receive the accelerator-timing signals directly, the

Xilinx Virtex-5 FPGA on the Timing Module will directly receive the signals via the daughter card instead from an ODM and optical receiver. Addition of the daughter card will also simplify the software from two separate applications that communicate via User Datagram Protocol (UDP) packets to one application that will reside on the timing card PC.

4.2 Firmware Design

In this section, the firmware design for the Timing Module and the optical communication card is described. In the firmware design, an interface module is designed in VHDL to interface the PCI Express endpoint core with the timing logic or the OCC logic described in the background section of the thesis.

4.2.1 Timing Module Design

The Timing Module design can be divided into four sections, as shown in Figure 4.3: endpoint block plus core, interface logic, timing logic, and TLK sync logic. The endpoint block plus core is generated via the CORE Generator utility of the Xilinx ISE tool. The core is configured and customized to match the Timing Module design. Customization of the core consists of four sections: basic parameter settings, base address registers (BAR) setting, configuration registers setting, and advanced settings. In the basic parameter settings, the lane width, interface frequency, ID initial values, and class code are specified. The base address register settings allow the user set the BAR address space. The configuration registers setting let the user set options for the device capabilities register and the link capabilities register. The advanced setting is left at the default setting in this design. Table 4.2 shows the settings used for the Timing Module design.

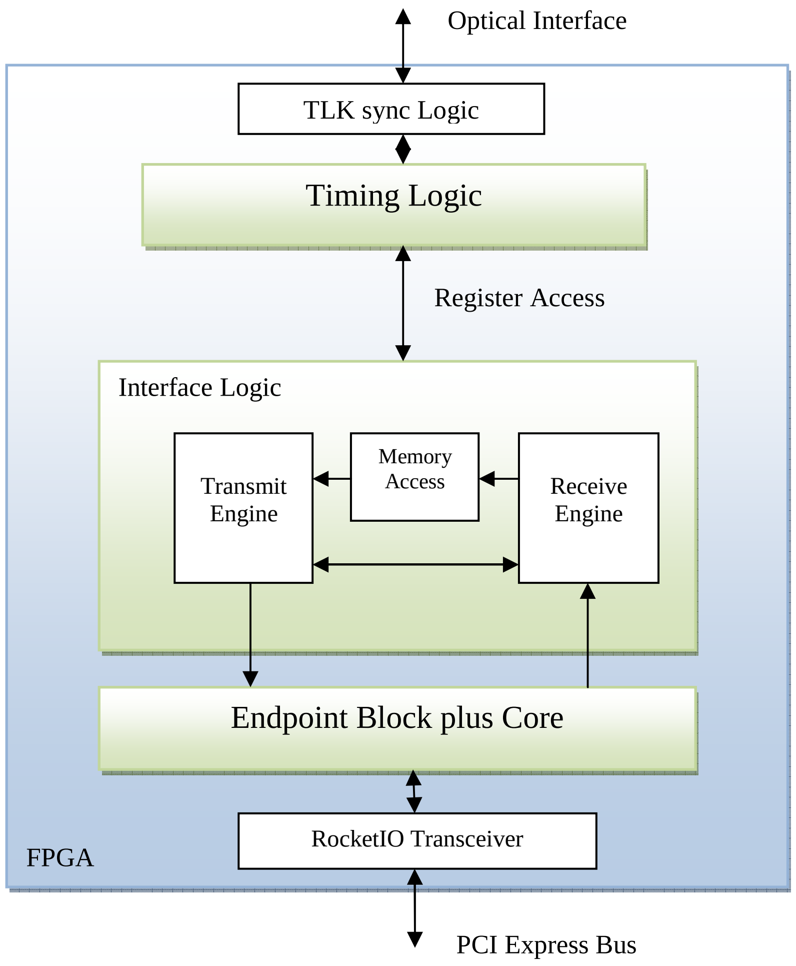


Figure 4.3: Timing Module firmware design

Lane Width	Reference frequency (MHz)	Base Address Registers (64 bit)	
		BAR0 (KB)	BAR2(KB)
x4	100	2	0

Table 4.2: Timing Module core customization settings

The interface logic is a VHDL module designed to interface the PCI Express core with the SNS timing logic. It also implements the interrupt generation. It can be divided into 3 major blocks: transmit engine, receive engine, and memory access. The transmit engine implements the transmission and the receive engine implements the reception of the PCI Express TLPs as discussed in Chapter 3. Packets sent to the core for transmission must follow the formatting rules for TLPs as specified in the PCI Express Base Specification [15]. The memory access unit utilizes block RAMs in the FPGA to store the PCI interface registers. Each PCI interface register is assigned a unique address in the BAR memory space. In case of a write request, the write data is stored at the incoming write address. In the case of a read request, the completion with data is fetched from the read address and sent via the transmit engine to the core. The timing logic configuration registers are also updated by the memory access module.

The transmit engine, as shown in Figure 4.4 is a state machine that transmits the outbound TLPs. The user logic is responsible for constructing the outbound packet whenever the completion of memory read request occurs. There are three states in the transmit engine: reset, transaction without complete, and transaction complete with data. In the reset state, the user logic asserts the transmit source ready (trn_tsrc_rdy_n) and start of frame (trn_tsof_n) transaction interface signals and presents the first QWORD when it is ready to transmit. Figure 4.5 [17] shows the operation of the outbound packet transmission. The core keeps the QWORD presented until the core asserts the destination ready signal (trn_tdst_rdy_n). Once the trn_tdst_rdy_n signal is asserted the QWORD is accepted immediately. If the read request is completion with data, the state will change to transaction complete with data, otherwise the state will be complete without data. In each of the transaction complete states, the user application keeps the trn_tsrc_rdy_n asserted and submits the next QWORD as shown in Figure 4.5.

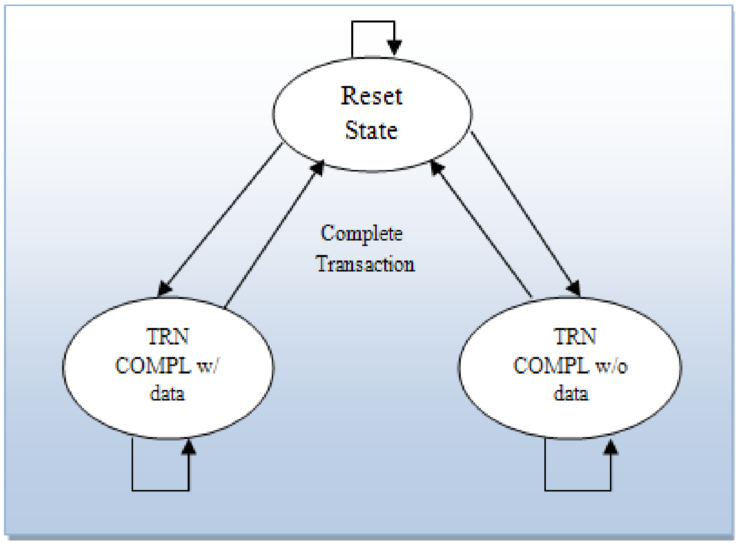


Figure 4.4: Transmit engine state machine

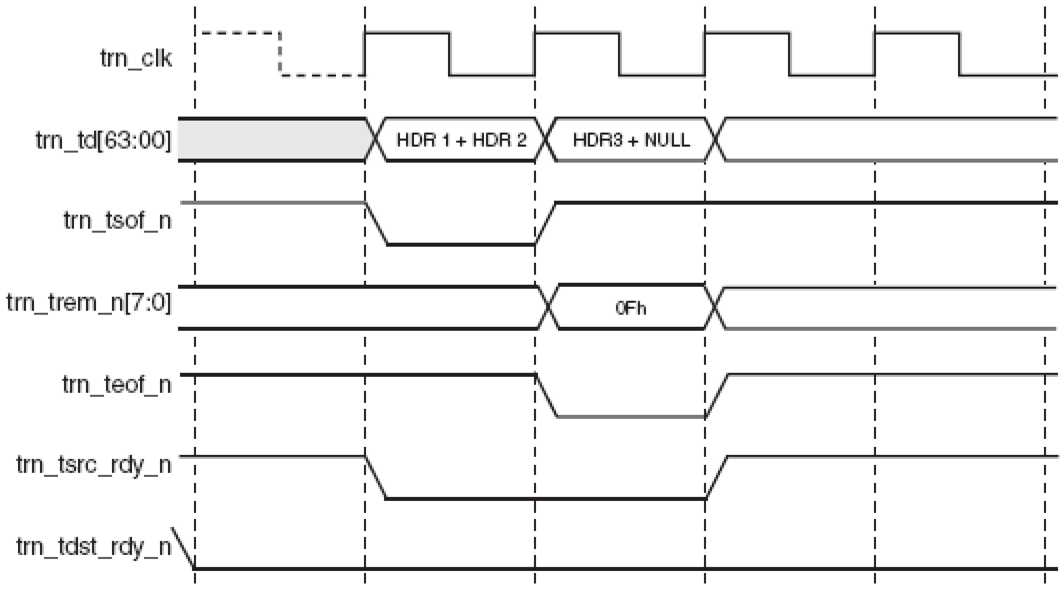


Figure 4.5: Outbound packet transmission

The receive engine, as shown in figure 4.6 has four states: reset, receive memory read TLP, receive memory write TLP, and wait. In the reset state, when the user logic is ready to receive the data, the user application asserts the receive destination ready transaction interface signal (trn_rdst_rdy_n) to indicate that it is ready to receive TLPs. The core asserts the receive source ready (trn_rsrc_rdy_n) and the receive start of frame (trn_rsof_n) signals to indicate that the user application is ready to receive the first TLP QWORD. Figure 4.7 [17] shows the waveforms describing the reception of the inbound packets. The TLP could be a read or write TLP request. The user application decodes the header information from the TLP to determine if it is a read or write request. If it is a write request, the state changes to the receive memory write TLP. When it is read request, the header information is sent to the transmit engine and the state is changed to the receive memory read TLP. The first and second DWORD of the memory TLP is decoded in both read and write TLP states. The state is then changed to the wait state and the state machine waits there until the write has been completed or the read has been successfully transmitted.

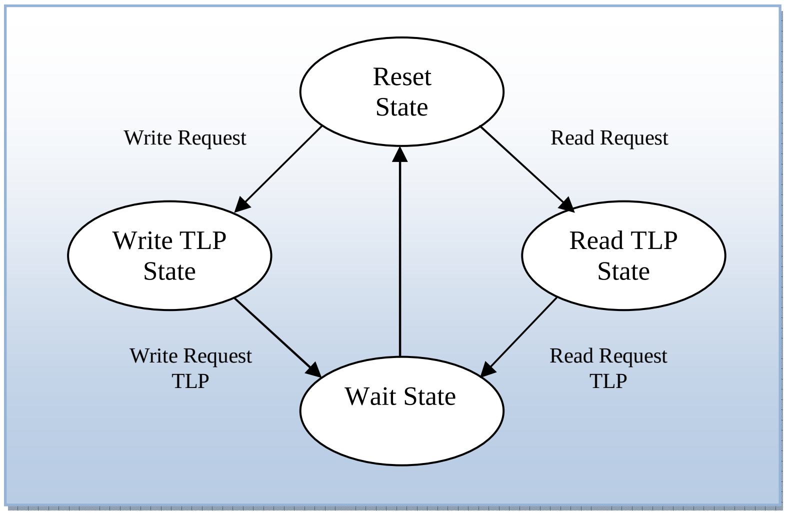


Figure 4.6: Receive engine state machine

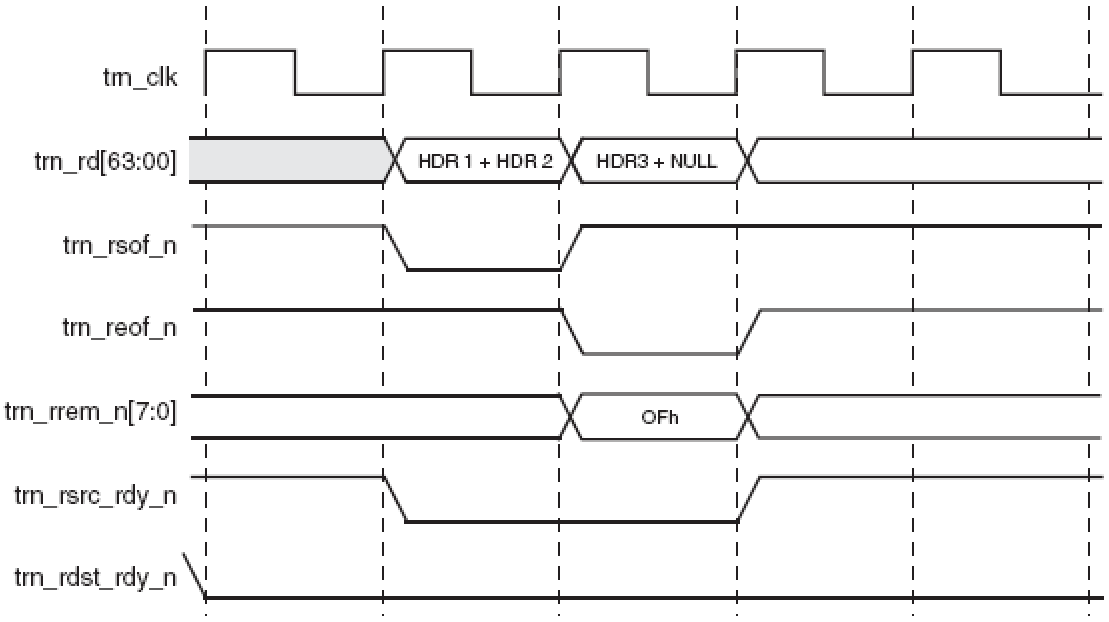


Figure 4.7: Inbound packet reception

The memory access module implements the Block RAM interface for the read and write data storage. Along with the data storage, it also generates and clears the interrupts by implementing several interrupt registers. The PCI Express interrupt generation is considerably more complex than that for the PCI-X interface. Legacy mode interrupt operation, as shown in the Figure 4.8 [17], has been implemented in the Timing Module. The user application first asserts cfg_interrupt_n and cfg_interrupt_assert_n signals to indicate the assertion of the interrupt. The user application also selects the legacy interrupt INTA using cfg_interrupt_di[7:0]. The core then asserts cfg_interrupt_rdy_n to indicate that an interrupt has been accepted. On the following cycle, the user application deasserts cfg_interrupt_n. After a certain period of time, the user application asserts cfg_interrupt_n and deasserts cfg_interrupt_assert_n to indicate deassertion of the interrupt. The core asserts cfg_interrupt_rdy_n to indicate that the interrupt deassertion has been accepted. On the following clock cycle user application deasserts cfg_interrupt_n.

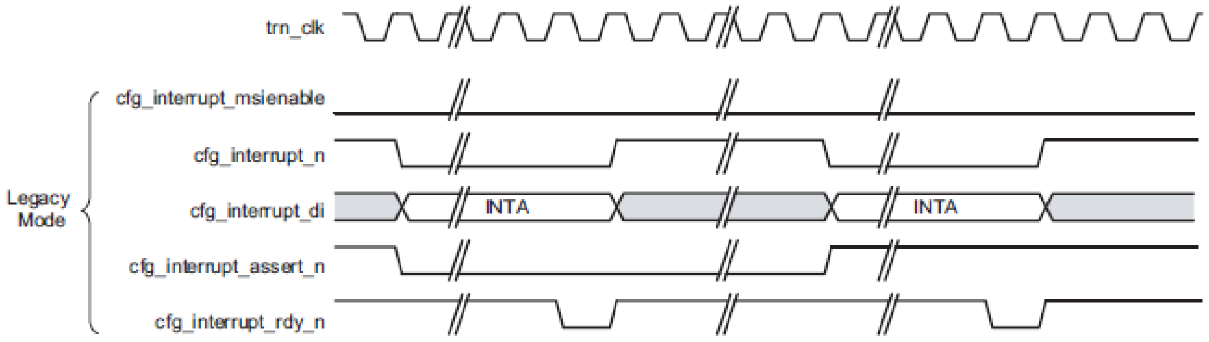


Figure 4.8: Legacy interrupt operation

Interrupts can be generated by the eight chopper vetoes, the beam veto, the PT0, Tstart pulse, or the Loss of lock pulse in the Timing Module design. The Tsync generation circuitry in the timing logic generates a Tsync signal if a chopper veto, PT0, or Tstart pulse is seen. The Timing Module interrupts the device driver if a Tsync pulse is asserted and updates all the read-only registers through the PCI Express bus. Therefore, the read-only registers are updated after every Tsync cycle ensuring fresh values from the instrument systems.

The interrupt module is designed as a state machine as shown in Figure 4.9 and has five states: intr reset, intr ACK, intr service, intr ACK2, and intr done. If an interrupt is requested, the cfg_interrupt_n and cfg_interrupt_assert_n signals are asserted in the intr reset state. Then the state changes to the intr ACK state. The state machine remains in the intr ACK state until the cfg_interrupt_rdy_n signal is asserted by the core to acknowledge the assertion of the interrupt. Once received, then the state is changed to the intr service state and the state machine then waits there for a random time until the interrupt has been serviced. The cfg_interrupt_n signal is asserted by the user logic to indicate deassertion of the interrupt. Once received, the state is then changed to intr ACK2. The state machine remains in the interrupt ACK2 state until the cfg_interrupt_rdy_n is asserted by the core to acknowledge deassertion of the interrupt. Both cfg_interrupt_n and cfg_interrupt_assert_n are deasserted and the state is changed to interrupt done state to indicate the completion of a single interrupt cycle as shown in Figure 4.9.

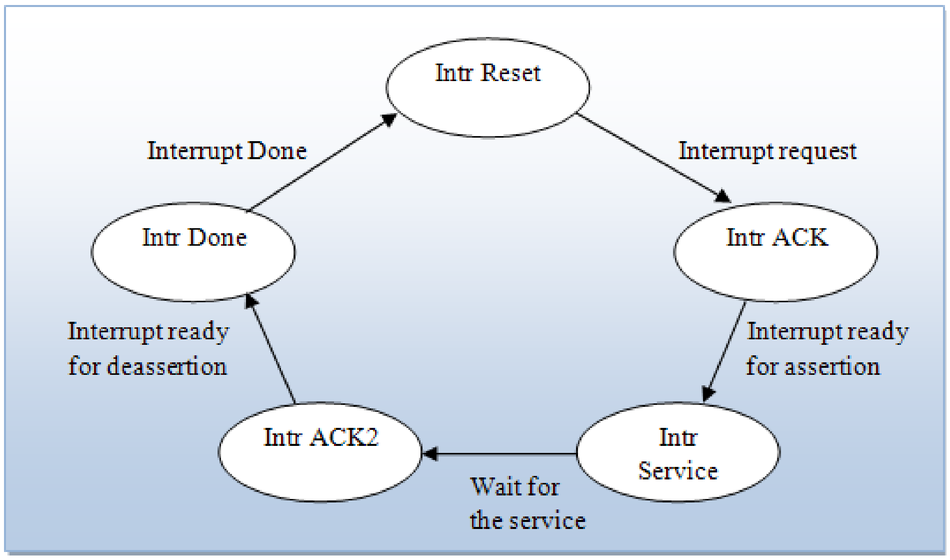


Figure 4.9: Interrupt generation state machine

The timing logic and the TLK sync blocks shown in Figure 4.3 are based on the previous PCI-X design. In the new design, we have added seven more status registers to verify the functionality of the Timing Module. We have also added four additional chopper control systems, meeting the total chopper control system originally planned for each Timing Module.

4.2.2 OCC Design

The OCC design is divided into four modules as shown in Figure 4.10: Endpoint Block Plus Core, Interface Logic, TLK Sync, and LVDS Sync. A simple MUX is implemented to select between the LVDS and the optical interface for data transfer. The endpoint block plus core is different from that for the Timing Module as it is customized for DMA transfer. In the BAR settings, the address space BAR0 is configured to be 64-bit memory with 4 KB aperture. The higher aperture is justified due to higher memory requirement for the DMA transfers in the OCC design. All other settings in the endpoint customization are the same as those for the Timing Module. Table 4.3 shows the settings used for the OCC design.

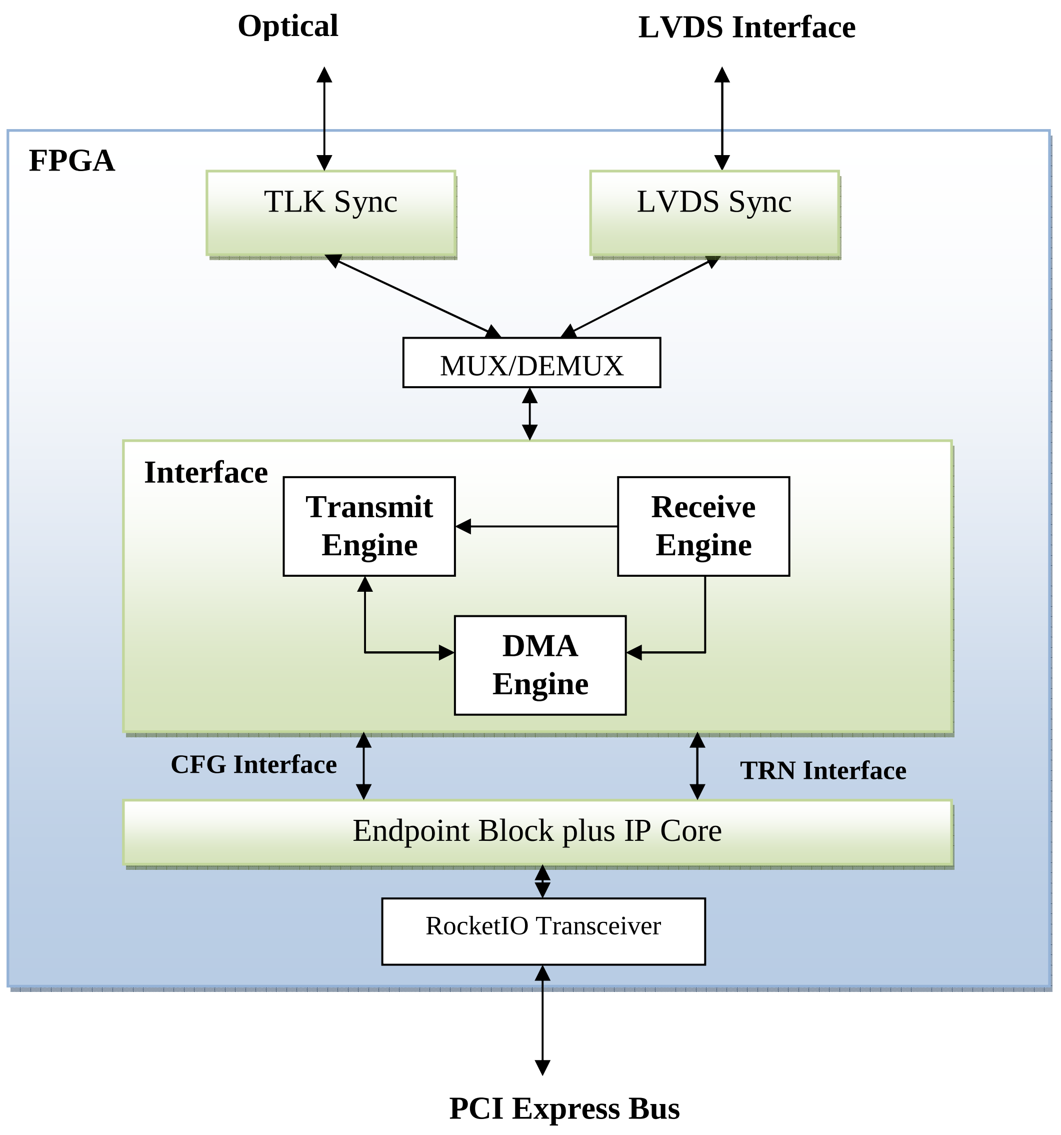


Figure 4.10: OCC firmware design

Lane Width	Reference frequency (MHz)	Base Address Registers (64 bit)	
		BAR0 (KB)	BAR2(KB)
x8	250	4	2

Table 4.3: OCC core customization settings

The receive engine is a state machine as shown in Figure 4.11. It has four states: reset, wait, TLP format type, and payload. The function of the receive engine is to process incoming TLPs and update the configuration register when the memory write requests are received. It also processes reads and transfers control to the transmit engine which generates the completion for read request. In the reset state, the receive engine decodes the received TLP header and handles the TLP according to the TLP format type. The TLP format type could be the read request, the write request, or the completion with the data TLP in response to the memory read request. In the wait state, the receive engine stalls and waits for a response from the other blocks, such as the transmit engine and the DMA engine, to satisfy its request. In the payload state, the receive engine receives the request completion with data TLPs and changes the state to the reset state when it receives all the data requested by the DMA engine.

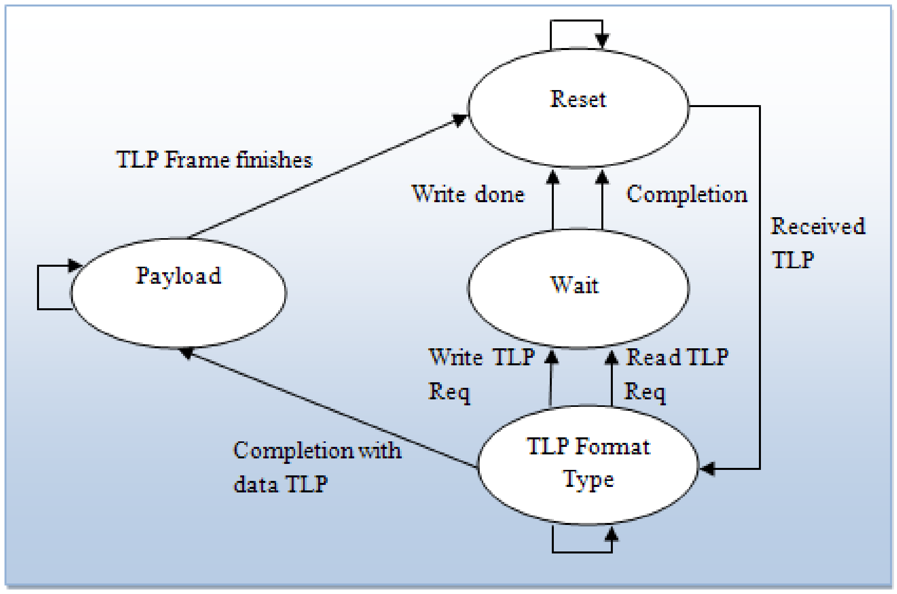


Figure 4.11: OCC receive engine

The transmit engine, as shown in Figure 4.12 has three states: reset, TLP format type, and payload. The transmit engine sends memory read and memory write requests to the root complex. The interrupt module is also a part of the transmit engine since the endpoint application must interrupt the driver after a successful requested transmission. The interrupt module generates legacy interrupts and has the same functionality as the Timing Module interrupt design. In the reset state, if a transmission request occurs, the DMA engine packs the TLP header and initiates the DMA transfer. Then the state changes to the TLP format type, which could be a simple read completion, memory write, or memory read. For a simple read, the state changes to the reset state after the completion with data. For memory read, the transmit engine sends the TLP with the read address and read count to the root complex (see Figure 3.1). The state changes to the reset state after sending the read request TLP. For memory write, the state machine changes to the payload state, where it sends out the write TLP packets until it reaches the expected count.

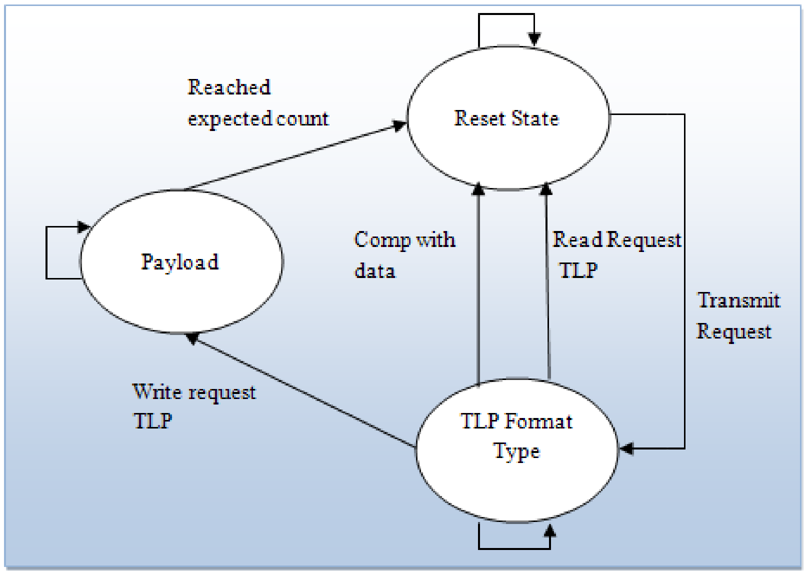


Figure 4.12: OCC transmit engine

The DMA engine initiates a DMA write or read operation. In OCC design, the DMA engine consists of status registers and configuration registers. Some operational information is required from the software application to configure the configuration registers. The DMA engine also implements the IDMA and ODMA memory areas as discussed in chapter 2. The Command Queue (CQ), Data Queue (DQ), and Inbound Message Queue (IMQ) for optical data transfer are also implemented. The software application sets up the configuration register to inform the DMA engine to initiate DMA write operations. The DMA engine generates memory write TLPs with the write DMA address, write DMA TLP size, and write DMA TLP count. Then the DMA engine writes either the LVDS data or optical interface data depending upon the configuration register settings. An interrupt is generated when the number of TLPs transferred matches the write DMA TLP count. When the DMA engine initiates a DMA read operation, it sends out a memory read TLP request to root complex via the transmit engine. The TLPs include read DMA TLP address, read DMA TLP size, and read DMA TLP count. The completion acknowledgement with data is received through the receive engine in response to the memory read request and stored in the IDMA memory area. Once the requested data size is received, the engine interrupts the driver through the transmit engine. The total payload received must match the read TLP count times the read DMA TLP size. There are several local registers implemented in the DMA engine to control the start and operation of DMA transfer initiation.

4.3 Software Systems

The software systems consist of the PCI driver and SNS user application. The PCI driver establishes a link between the user application and the Timing Module card or OCC. There are two PCI drivers designed by the SNS DAS group to communicate with the Timing Module and OCC respectively. Since both of the PCI-X drivers are compatible with the new PCI Express based design, the drivers provided the necesary basic operations for the new design without any modifications. The drivers are compiled with Microsoft visual studio 2005 and the appropriate communication parameters including device ID, vendor ID, and driver version were changed to make the driver completely compatible with the new designs. We were able to run the applications and perform read and write operations through the PCI Express bus with these drivers as will be discussed in the next chapter.

4.4 Summary

This chapter discussed the design and implementation of the new Timing Module and OCC. In the new PCB design, we provided the Virtex-5 FPGA schematic and pinout details. We also performed the FPGA schematic and other hardware verification. In the FPGA code modification, we designed the interface logics that interfaces the PCI Express core with the Timing Module and OCC user logic. We also discussed the state machines that are part of the interface logic. Chapter 5 discusses the resource utilization, performance estimation, and design verification of the new design.

CHAPTER FIVE

RESULTS

This chapter describes the results obtained from the design and implementations of the new PCI Express based Timing Module and Optical Communication Card (OCC). We discuss the FPGA resource utilization, the system performance analysis, and the verification of the PCI Express operations with the new designs. We also compare the results with the earlier PCI-X based Timing Module and OCC design.

5.1 Resource Utilization

5.1.1 Timing Module

The FPGA implementation of the earlier PCI-X based Timing Module requires 99% of the logic slices, 84% of the external IOBs, and 17% of the Block RAM on a Virtex-II XC2V1000 FPGA. Table 5.1 shows the resource utilization of the earlier design from the place and route report. One of the initial requirements of the Timing Module was to control eight chopper systems while presently it can control only four chopper systems due to logic area limitations. Implementing an additional four channels of the chopper controls would require 8% more logic slices in the Virtex-II device used in the original Timing Module card. As shown in Table 5.1 there is only 1% of logic slices remaining in the Virtex-II FPGA. In our new design, we use a Xilinx Virtex-5 XC5VLX50T FPGA, which has 3 times more logic slices to overcome the logic resource problem.

The new FPGA implementation for the PCI Express based Timing Module on the Virtex-5 including the PCI Express endpoint and Timing Module logic requires only 46%

of the total available logic slices, 34% of the external IOBs, and 16% of the Block RAM. The complete resource utilization of our design is shown in Table 5.2. Four additional channels of chopper control are implemented in the new design. With the new design and chopper control expansion, the Virtex-5 FPGA still has 50% of the logic space available for future expansion. The addition of a new daughter card and implementation of the associated transceiver logic will utilize this free logic space in the FPGA. The Virtex-5 device also provides twelve DCMs compared to only eight in Virtex-II FPGA, which will allow us to add the necessary DCMs for the daughter card reference clocks.

Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	5118	5120	99%
Number of Slice Flip Flops	4882	10240	47%
Number of LUTs	8043	10240	78%
Number of BRAMs	7	40	17%
Number of DCMs	5	8	62%
Number of IOBs	275	324	84%

Table 5.1: Virtex-II XC2V1000 utilization table for the Timing Module

Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	3102	7200	46%
Number of Slice Flip Flops	5863	28800	20%
Number of LUTs	6956	28800	24%
Number of BRAMs	10	60	16%
Number of DCMs	8	12	56%
Number of IOBs	167	480	34%

Table 5.2: Virtex-5 XC5VLX50T utilization table for the Timing Module

5.1.2 Optical Communication Card

Table 5.3 shows the resource utilization of the original OCC design. The PCI-X based implementation of the original OCC design requires 48% of the logic slices, 34% of the IOBs, and 41% of the Block RAM on a Virtex-II XC2V1500 FPGA. With only 48% of the logic slices used, the logic resources are not a major limitation in the OCC design. The Block Ram utilization in the design however is high (41%) due to large memory space implementation required for the PCI-X DMA transfer logic.

The new FPGA implementation of the PCI Express based OCC design requires 42% of the logic slices, 34% of the external IOBs, and 41% of the Block RAM available on the Virtex-5 XC5VLX50T FPGA. The complete resource utilization of the new OCC design is shown in Table 5.4. The PCI Express implementation of the OCC design also reveals that there is enough resources available in the FPGA for future logic expansion. The primary reason of Virtex-5 migration of OCC design is to implement PCI Express DMA transfer for higher throughput. The Block RAM utilization in the Virtex-5 device is still high due to increased memory requirements for the PCI Express DMA transfer compared to the old design. Virtex-5 provides more Block RAM memories compared to the Virtex-II device used in the original OCC design, so the absolute number of BRAMs is higher and the percentage of resources utilized remains roughly the same.

Logic Utilization	Used	Available	Utilization
Number of Slices	3757	7680	48%
Number of Slice Flip Flops	2487	15360	16%
Number of LUTs	5414	15360	35%
Number of BRAMs	20	48	41%
Number of DCMs	3	8	37%
Number of IOBs	184	528	34%

Table 5.3: Virtex II XC2V1500 utilization table for the OCC

Logic Utilization	Used	Available	Utilization
Number of Slices	3072	7200	42%
Number of Slice Flip Flops	5837	28800	20%
Number of LUTs	6526	28800	23%
Number of BRAMs	35	60	41%
Number of DCMs	8	12	66%
Number of IOBs	164	480	34%

Table 5.4: Virtex-5 XC5VLX50T utilization table for the OCC

5.2 Performance Estimation and Analysis

5.2.1 Timing Module

The maximum transfer rate of the PCI Express system is 2.5 GB/s per lane. This data rate is the raw bit transfer rate while the effective data transfer rate is much lower due to overheads and system design trade-offs. Table 5.5 [22] shows the link efficiency or the theoretical maximum data throughput with a maximum payload size of 128 bytes for the writes and reads.

Link Width	Write	Read
x1	1720	1520
x4	6880	6080
x8	13760	12160

Table 5.5: Theoretical bandwidth of different PCI Express link

The bandwidth of the PCI Express system is given by the following formula:

Bandwidth (BW) = [(Total Bytes transferred) ⁄ (Transfer Time)] x GB/s (1)

5.2.1.1 Write Bandwidth

In the Timing Module design, approximately 55-posted writes are transmitted at 256 bytes (Maximum Payload Size) each on x8 link. The total number of bytes transferred over the PCI Express link is given by:

Total bytes transferred = (number of posted writes) x (maximum payload size) (2)

Using Eqn. 2, the total bytes transferred for our system is 14080 bytes.

The transaction clock period is 4 ns. Therefore, the PCI Express x8 link can send 8 bytes every 4 ns. We assume a TLP with a 12-byte header with no Error Cyclic Redundancy Check (ECRC) and 8 bytes of address, results in 20 bytes of overhead. The transfer time of a single 32-bit addressable memory write is given by:

Write TLP transfer time = [(maximum payload + overhead) / 8 bytes / clock] x 4 ns/clock (3)

Using Eqn. 3, the write TLP transfer time is 138 ns for our system.

The Data Link Layer (DLL) also introduces traffic overhead to the total transfer time. The DLL packet transfer time is 4 ns. For every TLP sent, an acknowledgement (ACK) must be returned to the sender of the TLP to indicate successful receipt of the packet. Flow Control (FC) DLL packets are also sent by the link partner to indicate that the receiver has sufficient buffer space. The FC DLL packets are used to avoid receiver buffer overflow. We assume that the TLP to ACK ratio is one ACK for every five TLPs and an FC DLL packet is received every four TLPs. Therefore, the Total transfer time of all posted writes is given by [22]:

Total transfer time = (number of posted writes x write TLP transfer time) +

(number of ACKs x 4ns) + (number of FC DLL packets x 4 ns) *(4)*

Using Eqn. 4, the total write transfer time is 7686 ns for our system. Using Eqns. 1, 2, and 4, the total estimated write bandwidth in our case is obtained as 1.831 GB/s.

5.2.1.2 Read Bandwidth

There are a maximum of 10240 bytes of data reads on an x8 link considering a maximum read request size of 256 bytes and 40 status registers to be read at a time for the Timing Module. Therefore, the total number of bytes to be transferred over the PCI Express link is 10240 bytes. Here we assume 20 bytes of overhead for a memory read TLP. The memory read request does not contain any payload. The 32-bit addressable memory read TLP transfer time is given by:

Read TLP transfer time = [(overhead) / 8 bytes / clock] x 4 ns/clock *(5)*

Using Eqn. 5, the read TLP transfer time is 10 ns for our system.

The DLL packet transfer time is 4 ns. The majority of the read TLPs are 64 bytes. Therefore, the Read Completion Boundary (RCB) payload is assumed to be 64 bytes. The read completion transfer time is given by:

RCB transfer time = [(RCB payload + overhead) / 8 bytes / clock] x [4 ns / clock] *(6)*

Using Eqn. 6, the RCB transfer time is 42 ns for our system.

We assume that the TLP to ACK ratio is one ACK for every five TLPs and an FC DLL packet is received every four TLPs. We also assume the receiver incoming read to completion generation time is 300 ns [22]. Therefore the Total transfer time of the memory reads is given by:

Total transfer time = (number of memory reads) x [Read TLP transfer time + read completion generation time + 4 ns] + (RCB payload x RCB transfer time) + (overhead x 4ns) *(7)*

Using Eqn. 7, the total transfer time for 40 memory reads is 15328 ns for our system. Using Eqns. 1 and 7, the estimated total read bandwidth is calculated as 668 MB/s.

5.2.1.3 Comparison

The PCI Express implementation of the Timing Module has an estimated 1.832 GB/s write and 668 MB/s read bandwidth. The write bandwidth meets the desired 1 GB/s bandwidth requirement for the Timing Module operation. The read bandwidth of 668 MB/s is much lower than the theoretical bandwidth. The main reason is due to the assumption that the reads are not pipelined and can only process one read at a time. But the read bandwidth of the PCI-Express is higher that the PCI-X read bandwidth [27]. The graph in the Figure 5.1 shows the bandwidth comparison of the PCI-X, the x4 PCI Express, and the x8 PCI Express with respect to the Timing Module data payloads.

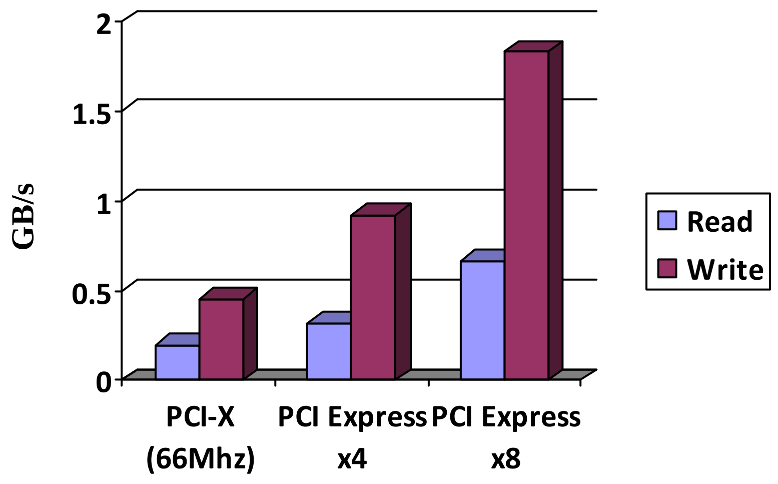


Figure 5.1: Performance comparison

5.2.2 Optical Communication Card

The OCC design has a higher bandwidth requirement compared to that of the Timing Module. To achieve the required higher data rate and performance, a bus master DMA is implemented to provide the desired data transfer rate for the OCC design. We studied the performance of the DMA transfer over the PCI Express bus using the new SNS PCI Express board with the Virtx-5 FPGA. We used the Xilinx Bus Master DMA (BMD) reference design [20] and Avnet software application [20] to study the performance of the system. We a used a 32 DWORD TLP size and a maximum payload size of 128 bytes to monitor the performance. We transferred 32 KB of data over x1, x4, and x8 links separately and studied the data rates over the PCI Express link, as shown in Figure 5.2. The data rates obtained are sufficient for the OCC design operation.

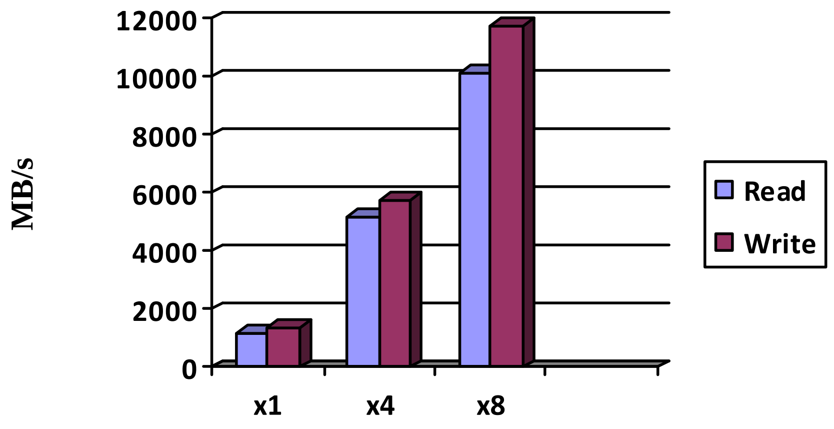


Figure 5.2: PCI Express DMA performance

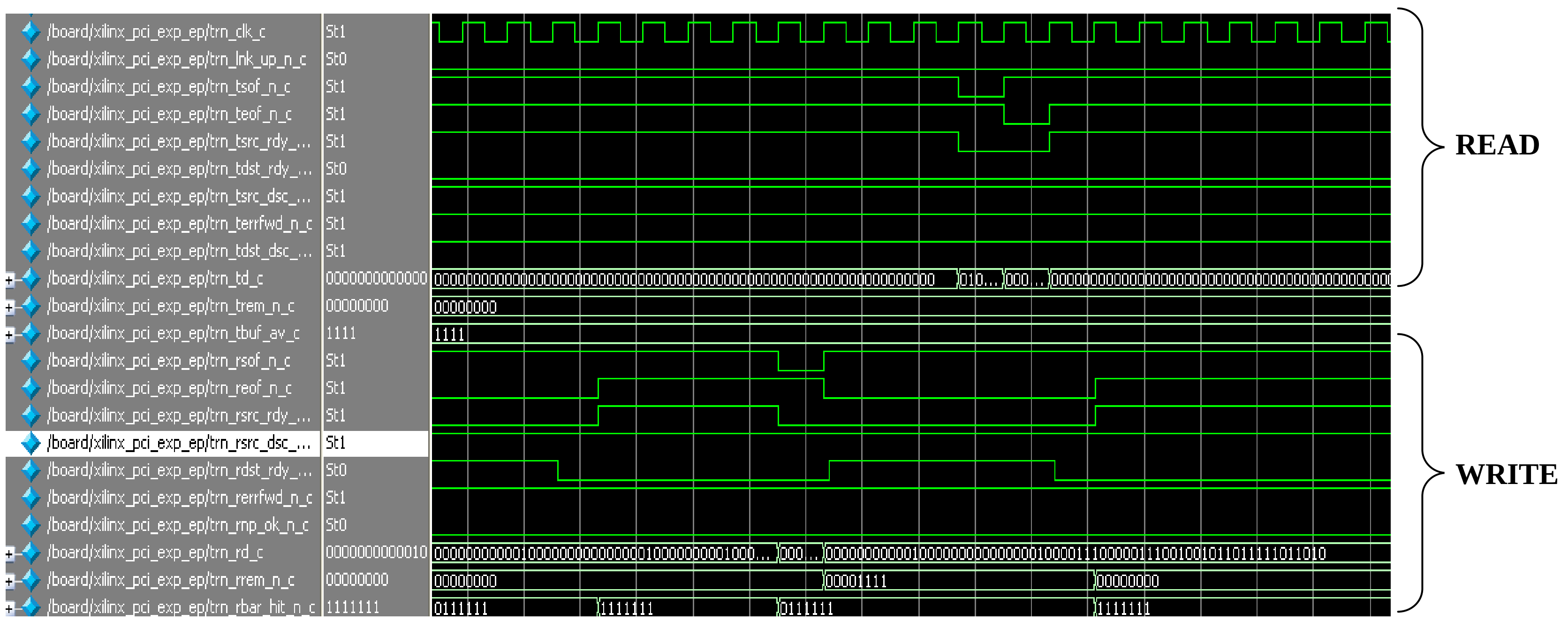
5.3 Design Verification

The Timing Module and OCC designs were verified by both functional simulation and real-time debug and verification tools. The functional simulations were performed using the Xilinx ISE 10.1 [23] and the ModelSim 6.5 SE simulators [24]. Real-time debug and verification tools such as the ChipScope Pro Core Generator, the ChipScope Pro Core Inserter, and the ChipScope Pro Core Analyzer [25] were also used to verify the design. The ChipScope Pro tool suite [26] enables real-time verification for FPGA designs by inserting low-profile soft debug cores into the design or netlist. The ChipScope Pro Analyzer tool is a logic and bus analysis interface that enables and sets the trigger conditions to show the real-time signals from the FPGA.

5.3.1 Timing Module Verification

We simulated the PCI Express functionalities when integrated with the Timing Module user logic to show the basic read and write operations over the PCI Express bus.

Figure 5.3 shows the simple read and write operations of the PCI Express protocol simulation. The trn_lnk_up_c signal assertion indicates that the PCI Express link is active. Whenever there is a target write, the trn_rsof_n_c signal is asserted and the trn_rd_c receives the TLP data for the target write. Assertion of the trn_reof_n_c signal indicates the end of the write transaction. The simulation also shows the target completion with data TLP. The real time read and write operations of the PCI Express protocol are also verified on the FPGA using ChipScope pro Analyzer. Figure 5.4 shows a Timing Module read completion with data packet. The trn_td bus contains valid packet header and data between the assertion of trn_tsof_n and trn_teof_n signals. When the PCI Express core is ready to receive the data, it asserts trn_tsrc_rdy_n signal and the packet is transmitted. Figure 5.5 shows the PCI Express write operation. The data header and the payload is received via trn_rd. Assertion of trn_rsof_n indicates the reception of the header packet and assertion of trn_esof_n indicates the completion of one write operation. The PCI Express core asserts trn_rsrc_rdy_n signal to indicate transmission of the TLP.



```
/board/xilinx_pci_exp_ep/trn_clk_c          St1
/board/xilinx_pci_exp_ep/trn_lnk_up_n_c     St0
/board/xilinx_pci_exp_ep/trn_tsof_n_c       St1
/board/xilinx_pci_exp_ep/trn_teof_n_c       St1
/board/xilinx_pci_exp_ep/trn_tsrc_rdy_...   St1
/board/xilinx_pci_exp_ep/trn_tdst_rdy_...   St0
/board/xilinx_pci_exp_ep/trn_tsrc_dsc_...   St1
/board/xilinx_pci_exp_ep/trn_terrfwd_n_c    St1
/board/xilinx_pci_exp_ep/trn_tdst_dsc_...   St1
/board/xilinx_pci_exp_ep/trn_td_c           0000000000000
/board/xilinx_pci_exp_ep/trn_trem_n_c       00000000
/board/xilinx_pci_exp_ep/trn_tbuf_av_c      1111
/board/xilinx_pci_exp_ep/trn_rsof_n_c       St1
/board/xilinx_pci_exp_ep/trn_reof_n_c       St1
/board/xilinx_pci_exp_ep/trn_rsrc_rdy_...   St1
/board/xilinx_pci_exp_ep/trn_rsrc_dsc_...   St1
/board/xilinx_pci_exp_ep/trn_rdst_rdy_...   St0
/board/xilinx_pci_exp_ep/trn_rerrfwd_n_c    St1
/board/xilinx_pci_exp_ep/trn_rnp_ok_n_c     St0
/board/xilinx_pci_exp_ep/trn_rd_c           0000000000010
/board/xilinx_pci_exp_ep/trn_rrem_n_c       00000000
/board/xilinx_pci_exp_ep/trn_rbar_hit_n_c   1111111
```

Figure 5.3: Timing module PCI Express simulation

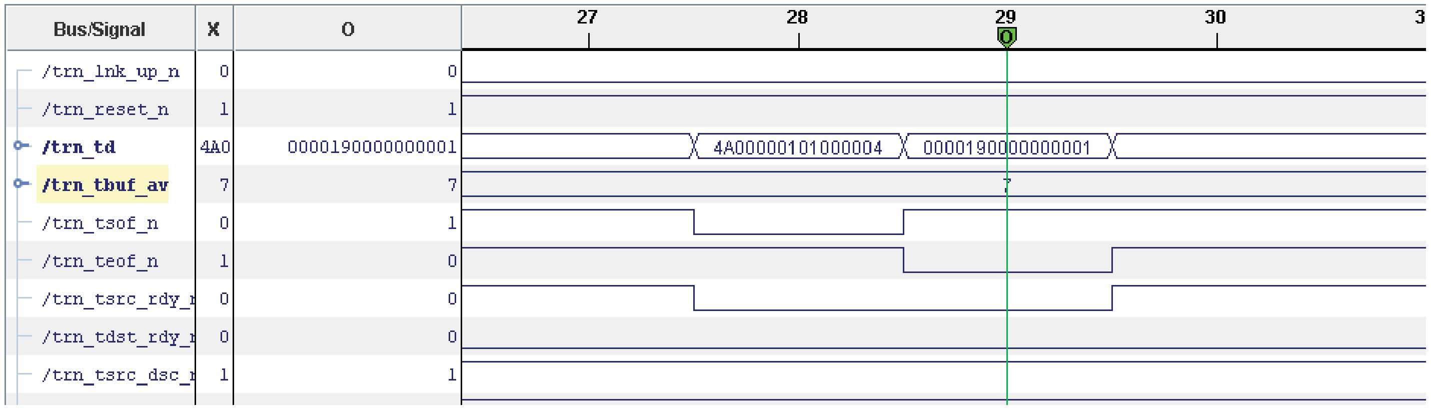


Figure 5.4: Timing module read operation

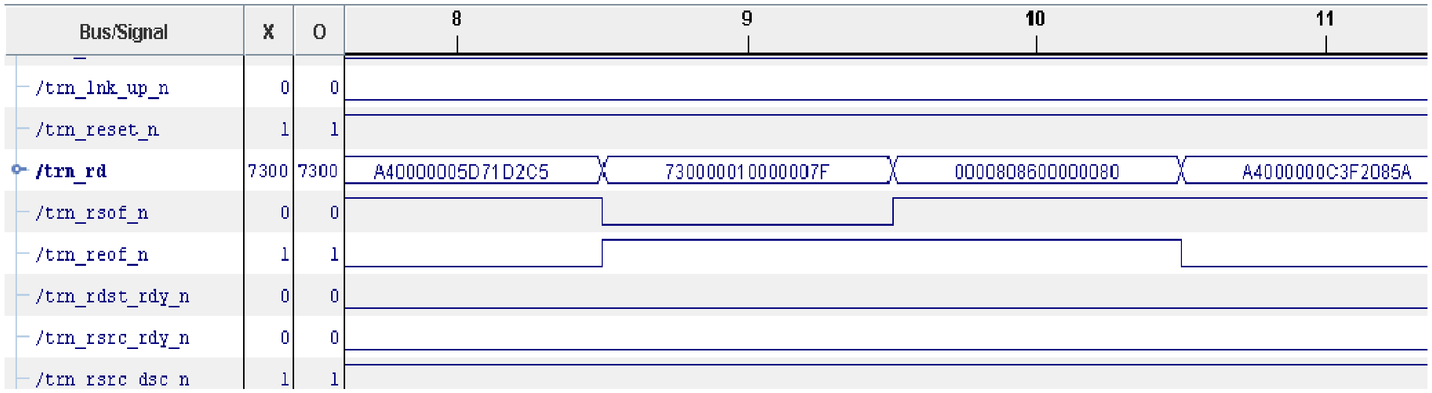


Figure 5.5: Timing module write operation

5.3.2 OCC Verification

We also simulated the DMA memory burst for the OCC design using ModelSim 6.5 SE simulator. The simulation waveform in Figure 5.6 shows the DMA transfer from the OCC to the system memory. When the OCC design is ready to transfer data to the system memory, it asserts the wr_dma_start signal. The wr_dma_done signal is asserted to indicate that the dma_wr_count value matches the number of TLPs to be transferred. Figure 5.7 shows the DMA transfer from the system memory to the OCC. The rd_dma_start signal is asserted to initiate a read DMA transfer. When the dma_rd_count value matches the number of the TLPs to be transferred, the DMA read is completed with the assertion of the rd_dma_done signal.

The DMA transfers were also verified on the OCC system using the ChipScope pro logic analyzer. Figure 5.8 shows the 16 TLPs being transferred from the OCC to the system memory and Figure 5.9 shows the burst transfer from the system memory to the OCC.

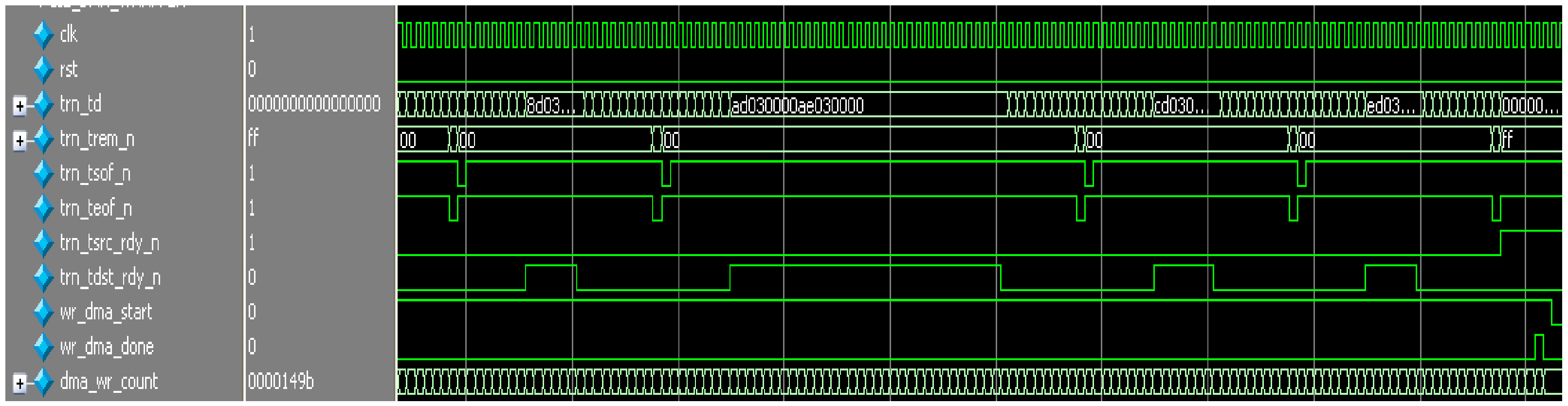


Figure 5.6: Simulation of DMA transfer from OCC to system memory

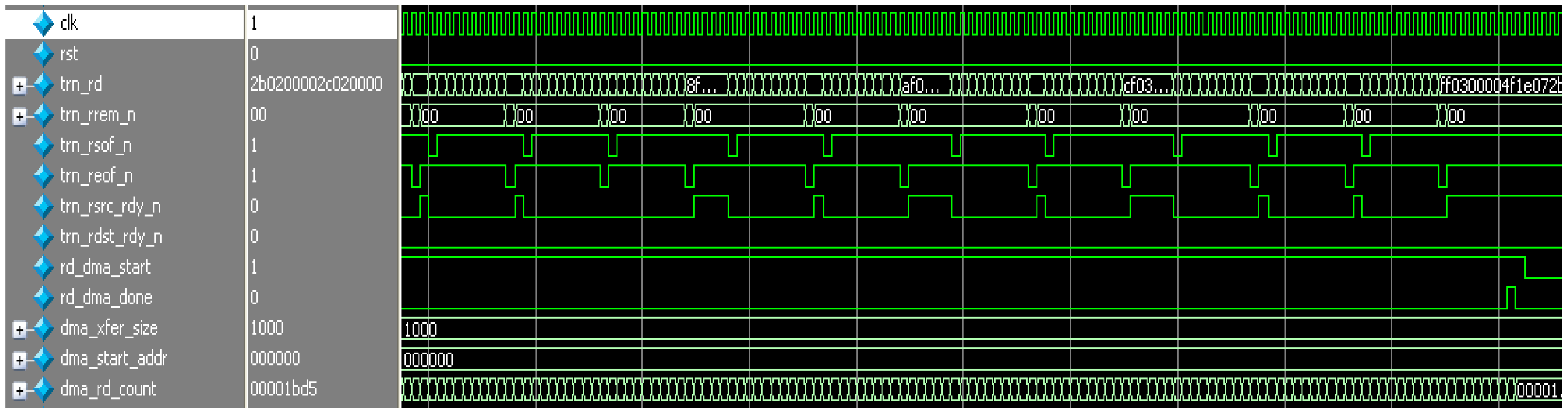


Figure 5.7: Simulation of DMA transfer from system memory to OCC

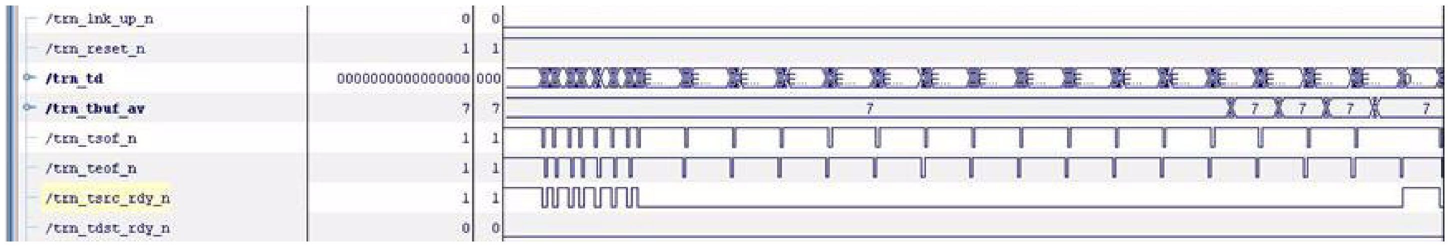


Figure 5.8: DMA transfer from OCC to system memory

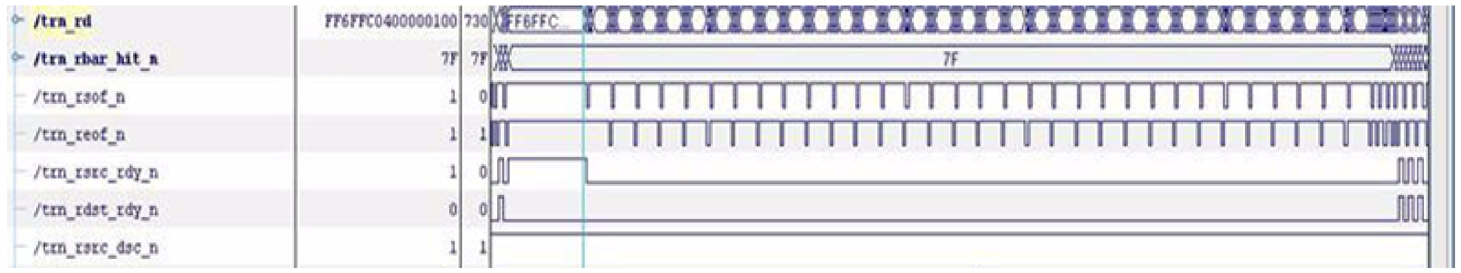


Figure 5.9: DMA transfer from system memory to OCC

5.3.3 Interrupt Verification

The PCI Express interrupt module was also simulated to verify the operation of the interrupt generation. The waveforms in Figure 5.10 shows the PCI Express legacy interrupt generation. It also shows the five states of operation that completes a single interrupt. As shown in Figure 5.10, the interrupt module state machine enters the intr_rst state and asserts cfg_interrupt_n and cfg_interrupt_assert_n signals to indicate the interrupt generation. Then the state changes to the intr_ack state. When the user application receives the cfg_interrupt_rdy_n signal from the core, it de-asserts the cfg_interrupt_n signal and changes the state to the intr_srvc state. The state machine remains in the intr_srvc state for a random period until the interrupt is serviced and then asserts the cfg_interrupt_n signal again. The state changes to the intr_ack2 state and waits there for the cfg_interrupt_rdy_n signal from the core. Once the cfg_interrupt_rdy_n signal is asserted by the core, the state changes to the intr_done state to indicate the completion of a single legacy interrupt operation.

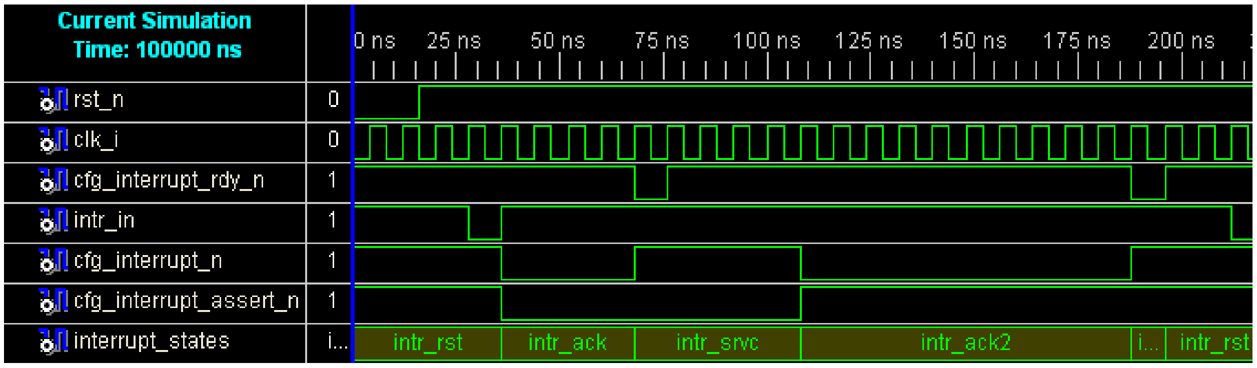


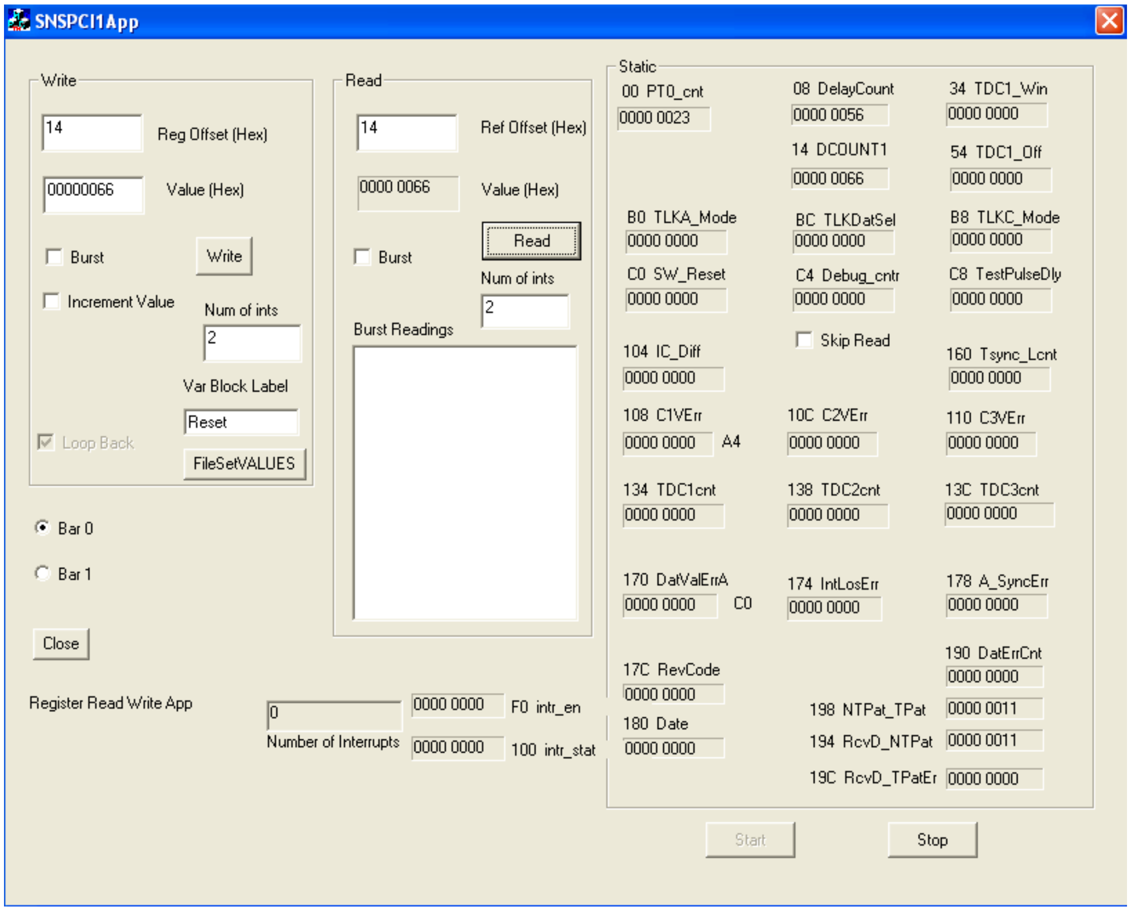
Figure 5.10: Interrupt generation simulation

5.3.4 LVDS and Optical Transceiver Loopback

Apart from the verification of the PCI Express integration, the optical transceiver loopback tests were also performed to verify the operation of the optical transceiver hardware on the board. We designed a diagnostic test bench in VHDL using Block RAM for the loopback data source to verify the optical transceiver hardware. For the optical tranceivers, we sent random data for transmission over the optical link. When the transmission is complete we wait for the reception of data to complete. If the transmitted data and the received data matches, it verifies the integrity of the optical transceiver hardware.

5.3.5 SNS Application and Driver Testing

We used the SNS PCI Express driver and application to write and read from the timing registers. The driver is a kernel mode PCI driver designed in the Windows operating system by SNS DAS group. We re-compiled the driver in Windows Visual Studio 2005 and changed the device parameters to work with the new PCI Express based PCB. Figure 5.11 shows a screenshot of the SNS application that is used to read and write to the configuration registers.


SNSPCI1App

Write
Reg Offset (Hex): 14
Value (Hex): 00000066
Burst
Write
Increment Value
Num of ints: 2
Var Block Label: Reset
[✓ Loop Back]
FileSetVALUES

[● Bar 0]
Bar 1
Close

Read
Ref Offset (Hex): 14
Value (Hex): 0000 0066
Read
Burst
Num of ints: 2
Burst Readings

Register Read Write App
Number of Interrupts: 0
F0 intr_en: 0000 0000
100 intr_stat: 0000 0000

Static

00 PT0_cnt: 0000 0023	08 DelayCount: 0000 0056	34 TDC1_Win: 0000 0000
	14 DCOUNT1: 0000 0066	54 TDC1_Off: 0000 0000
B0 TLKA_Mode: 0000 0000	BC TLKDatSel: 0000 0000	B8 TLKC_Mode: 0000 0000
C0 SW_Reset: 0000 0000	C4 Debug_cntr: 0000 0000	C8 TestPulseDly: 0000 0000
104 IC_Diff: 0000 0000	Skip Read	160 Tsync_Lcnt: 0000 0000
108 C1VErr: 0000 0000 A4	10C C2VErr: 0000 0000	110 C3VErr: 0000 0000
134 TDC1cnt: 0000 0000	138 TDC2cnt: 0000 0000	13C TDC3cnt: 0000 0000
170 DatValErrA: 0000 0000 C0	174 IntLosErr: 0000 0000	178 A_SyncErr: 0000 0000
17C RevCode: 0000 0000		190 DatErrCnt: 0000 0000
180 Date: 0000 0000	198 NTPat_TPat	0000 0011
	194 RcvD_NTPat	0000 0011
	19C RcvD_TPatEr	0000 0000

Start
Stop

Figure 5.11: SNS software application

5.4 Summary

The Timing Module and the OCC designs were verified using the functional simulations and in-system debugging methods. The Timing Module and the OCC designs were verified with the appropriate data from the function generators implemented in the designs. As a part of the future work, the designs will be tested in the real-time environment with Optical Ditribution Module (ODM) data at SNS. Chapter 6 will conclude the thesis with the discussion of possible future work for this project.

CHAPTER SIX

CONCLUSION

In this thesis, we have presented the implementation of the PCI Express in the SNS Timing Module and Optical Communication Card (OCC) designs. The PCI Express is a standard interface for the next generation of embedded applications due to its high bus throughput. The PCI-X bus interface used in the original Timing Module and OCC designs does not provide the required data rates due to its limited bandwidth. Therefore, the PCI Express bus was selected to replace the PCI-X bus in the Timing Module and OCC designs. The new designs also use a much larger FPGA to provide more functionality and room for future expansion.

In chapter 2, we presented the theory of operation of the earlier Timing Module and OCC design. We studied the operation of the original designs to modify the PCI interface and allow it to work with the PCI Express bus. We also presented all of the timing signals and registers that are required for the Timing Module and OCC operation.

Chapter 3 discussed the detailed PCI Express architecture and topology. We described the PCI Express protocol and how it works with the PCI Express endpoint core. The PCI Express core is more complex than the existing PCI-X core due to the implementation of three additional protocol layers. We also discussed the transaction layer packets (TLP) of the PCI Express protocol, which are responsible for the transmission of data between the endpoint core and the user application. We finally examined the Programmed I/O (PIO) and the Bus Master DMA (BMD) designs from Xilinx to understand the PCI Express TLP processing and the DMA transfers.

In Chapter 4, we presented the detailed design and implementation methodologies. For the Timing Module, we designed the receive engine, the transmit engine, the memory access module, and the interrupt module to interface the timing logic with the PCI Express core. For the OCC, we designed the receive engine, the transmit engine, and the DMA engine to initiate DMA transfers. Additional functionality will be added in the future for the complete operation of the OCC design.

Chapter 5 discussed some of the results we obtained from our design and implementation. The earlier Timing Module consumes 99% of the logic slices of the Xilinx Virtex-II FPGA. Migration of the FPGA design to a Xilinx Virtex-5 provided enough logic to meet current requirements and allow expansion of the design in the future. The Virtex-5 FPGA also provided a PCI Express hard IP core, which simplifies the design by creating a wrapper around the core. We also explained the achievable bandwidth for the PCI Express bus for both non-DMA and DMA transfers. The bandwidth analysis shows that the PCI Express bus provides much higher bandwidth compared to the earlier PCI-X bus. We also verified our design using VHDL simulators and real time JTAG signal captures. Testing with the real time data from the instrument systems still remains for both designs. It is important to note that the designs can only be tested at SNS with real-time signals from Optical Distribution Modules (ODM). We tested our designs using the data generated from the function generators implemented within the FPGA design. We also used the SNS PCI Express software application and drivers to verify the PCI Express operation and compatibility.

This work has yielded the following contributions to the design of the new Timing Module and OCC for the SNS instrument systems.

- FPGA related circuit design for the new PCB that functions as both the Timing Module and OCC board
- Schematic verification and hardware testing
- Simulation of the PCI Express core using ModelSim 6.5 SE simulators
- VHDL code implementation for the Timing Module PCI Express interface
- VHDL code implementation help for the OCC PCI Express DMA design
- Hardware debugging and troubleshooting using ChipScope Pro tools and on-board LEDs

The future work for this project involves testing the Timing Module and OCC design in the real-time environment at SNS. We will also add the daughter card to the system eliminating the need for the Accelerator Timing Card (ATC). To simplify the Printed Circuit Board design for the daughter card, we can replace the TLK2501 [14] transceivers currently used on the ATC with the in-build RocketIO transceivers on the Virtex-5 FPGA. The transceiver logic can then reside on the Virtex-5 FPGA, which has ample remaining logic resources for the implementation. The work will involve implementation of the RocketIO GTP cores on the Virtex-5 FPGA with interface logic. The bandwidth provided by the Virtex-5 RocketIO GTP is higher than that of the TLK2501 transceivers and therefore sufficient for our requirements.

There are 24 instrument systems installed in the SNS facility at Oak Ridge National laboratory. In many of the instrument systems, the transfer rate requirement is at least 1GB/s to collect all neutron events. Redesign of the Timing Module and OCC to incorporate the Virtex-5 FPGA and PCI-Express interface not only provides the necessary throughput but also provides enough logic area for future expansion as required.

REFERENCES

[1] Spallation Neutron Source, Data Acquisition Systems,
http://neutrons.ornl.gov/instrument_systems/components/das.shtml, Aug 20, 2009.

[2] Spallation Neutron Source, Timing Module,
http://neutrons.ornl.gov/instrument_systems/components/timing.shtml, Aug 20, 2009.

[3] Spallation Neutron Source, Instrument Systems Timing Module Technical Reference Manual, February, 2007.

[4] Spallation Neutron Source, Optical Communications Board,
http://neutrons.ornl.gov/instrument_systems/components/optical_comm.shtml, Aug 20, 2009.

[5] Spallation Neutron Source, Instrument Systems Optical Communication Card Technical Reference Manual, Dec. 2004.

[6] Xilinx Endpoint Block Plus Wrapper for PCI Express,
http://www.xilinx.com/products/ipcenter/V5_PCI_Express_Block_Plus.htm, Nov. 2009.

[7] A Low-Cost PCI Express Solution,
http://www.xilinx.com/publications/solguides/be_01/xc_pdf/p07-08_be1-pci.pdf, Sep. 2005.

[8] Spallation Neutron Source, Neutron Choppers,
http://neutrons.ornl.gov/instrument_systems/components/chopper/index.shtml, Aug 20, 2009.

[9] Spallation Neutron Source, Detectors,
http://neutrons.ornl.gov/instrument_systems/components/detector.shtml, Aug 20, 2009.

[10] LogiCORE IP Initiator/Target v5.166 for PCI-X, User Guide,
http://japan.xilinx.com/support/documentation/ip_documentation/pcix_64_ug160.pdf, Sep 19, 2008.

[11] Spallation Neutron Source, LPSD ROC Board,
http://neutrons.ornl.gov/instrument_systems/components/roc.shtml, Aug 20, 2009.

[12] Spallation Neutron Source, FEM Board,

http://neutrons.ornl.gov/instrument_systems/components/fem.shtml, Aug 20, 2009.

[13] Spallation Neutron Source, DSP Board,

http://neutrons.ornl.gov/instrument_systems/components/dsp.shtml, Aug 20, 2009.

[14] TLK2501 1.5 TO 2.5 GBPS Transceiver

http://www.datasheetcatalog.org/datasheet/texasinstruments/tlk2501.pdf, Feb. 2002.

[15] PCI Express Base Specification Revision 1.0, April 29, 2002.

[16] Creating a PCI Express™ Interconnect,

http://www.pcisig.com/specifications/pciexpress/resources/PCI_Express_White_Paper.pdf,

Nov. 2009

[17] LogiCORE™ IP Endpoint Block Plus v1.12 for PCI Express, UG341 September 16, 2009.

[18] Virtex-5 FPGA RocketIO GTP Transceiver, UG196 (v2.0) June 10, 2009.

[19] LogiCORE™ IP Endpoint Block Plus v1.9 for PCI Express, UG343 September 19, 2008.

[20] Bus Master DMA Reference Design for the Xilinx Endpoint Block Plus Core for PCI Express, XAPP1052 (v1.1) August 22, 2008.

[21] Virtex-5 FPGA User Guide,

http://www.xilinx.com/support/documentation/user_guides/ug190.pdf, Nov 5, 2009.

[22] Virtex-4 FPGA User Guide,

http://www.xilinx.com/support/documentation/user_guides/ug070.pdf, Dec 1, 2008.

[23] ISE 10.1 In-Depth Tutorial,

http://www.xilinx.com/direct/ise10_tutorials/ise10tut.pdf, Nov. 2009.

[24] ModelSim SE User’s Manual,

http://portal.model.com/modelsim/resources/references/modelsim_se_user.pdf, Nov. 2009.

[25] ChipScope Pro 10.1 Software and Cores User Guide,

http://www.xilinx.com/ise/verification/chipscope_pro_sw_cores_10_1_ug029.pdf, March 24, 2008.

[26] ChipScope Pro 10.1 Serial I/O Toolkit User Guide

http://www.xilinx.com/support/documentation/sw_manuals/chipscope_pro_siotk_10_1_ug213.pdf, Mar 24, 2008.

[27] Approaching PCI Bandwidth with FPGA,

http://www.nalanda.nitc.ac.in/industry/appnotes/xilinx/documents/products/logicore/pci/docs /performa.pdf, Nov. 2009.