ABSTRACT

Design and Automated Testing of PCI Express Interface of
Proton Computed Tomography Detectors

Yu Yang, M.S.E.C.E.

Mentor: Keith Evan Schubert, Ph.D.

Throughout this thesis, I will propose a transmit-received-engine based logic design proposed by this thesis works at the PCI Express Transaction Layer in collaboration with Xilinx 7 Series FPGAs Integrated Block for PCI Express. By automated testing and results evaluation, the new design can speed up the original Ethernet link speed by a factor of 30, At the same time, supports the needs of the new signal peaks in 50 ns. Therefore, two key concerns of the existing Phase-II pCT scanner hardware upgrade can be satisfied.

Design and Automated Testing of PCI Express Interface of
Proton Computed Tomography Detectors

by

Yu Yang, B.E., M.E.

A Thesis

Approved by the Department of Engineering and Computer Science

_____

Kwang Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of

Master of Science in Electrical and Computer Engineering

Approved by the Thesis Committee

_____

Keith Evan Schubert, Ph.D., Chairperson

_____

Robert J. Marks II, Ph.D.

_____

Young-Rae Cho, Ph.D.

Accepted by the Graduate School

May 2019

_____

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES ix
CHAPTER ONE 1
Introduction 1
Background 1
Organization of Thesis 3

CHAPTER TWO 5
PCI Express Specification Introduction 5
PCI Express Link 5
PCI Express Architecture 6
PCI Express Layering 8

CHAPTER Three 11
Transaction Layer Specification 11
Transaction Types and Address Spaces 11
Packet Format Overview 13
Packet Definition 14

CHAPTER FOUR 24
Introduction to Xilinx 7 Series FPGAs Integrated Block for PCI Express 24
Transaction Interface 25
Credit-Based Core Buffering/Flow Control 32

CHAPTER FIVE 33
A Transmit-Receive Engine Based Logic Design 33
Tx Engine and FWFT FIFO Choice 35
Rx Engine 46

CHAPTER SIX 49
Automated Test and Performance Evaluation 49
Design Testing 49
Performance Analysis 61

CHAPTER Seven 64
Conclusion 64
Summary 64
Future Directions 65

BIBLIOGRAPHY 67

LIST OF FIGURES

Figure 1.1. The Phase-II pCT scanner in the Northwestern Medicine Chicago Proton Center 2

Figure 2.1. PCI Express Link 5

Figure 2.2. Fabric Example 7

Figure 2.3. PCI Express Layering 8

Figure 2.4. Packet Flow Through Layers 9

Figure 3.1. TLP Format 13

Figure 3.2. Byte 0 of TLP Header 15

Figure 3.3. 64-bit Format Routing 18

Figure 3.4. 32-bit Format Routing 18

Figure 3.5. 1st /Last DW BEs 19

Figure 3.6. Transaction Descriptor 20

Figure 3.7. Transaction ID 21

Figure 3.8. Attributes Field 21

Figure 4.1. Top Level Functional Blocks and Interfaces 25

Figure 4.2. Transmit Interface 26

Figure 4.3. Receive Interface 28

Figure 4.4. A Subset of Configuration Interface 30

Figure 4.5. Memory 32 TLP on AXI4 Interface 30

Figure 4.6. 3 DW TLP with Payload 31

Figure 4.7. 4 DW Header with Payload 31
Figure 5.1. Top Level Hierarchy 33
Figure 5.2. Tx Engine State Diagram 36
Figure 5.3. Destination Throttle 37
Figure 5.4 Source Throttle 38
Figure 5.5. Mixed Mode Throttle 38
Figure 5.6. Standard FIFO Timing 39
Figure 5.7. Standard FIFO Timing 40
Figure 5.8. An Edge Case of Standard FIFO 40
Figure 5.9. FWFT FIFO Timing 42
Figure 5.10. FWFT FIFO Timing with Throttle 42
Figure 5.11. An Edge Case of FWFT FIFO usage 43
Figure 5.12. Tradition FSM Design for Tx Engine 44
Figure 5.13. A Counter Based FSM Design with Marco-Defined Parameters 46
Figure 5.14. Rx Engine State Diagram 48
Figure 6.1. PCI Express Structure in PC 50
Figure 6.2. Test Environment Structure 51
Figure 6.3. Root Complex Model Structure 52
Figure 6.4. Test Flow 53
Figure 6.5. Simulate Log Example 55
Figure 6.6. Error Log Example 55

Figure 6.7. Type 0 Configuration TLP Write Tx Log 55

Figure 6.8. Automated Testing Batch Program Code Example..... 57

Figure 6.9. 128 DW Memory 32 Read Request Error Timing 58

Figure 6.10. Device Control/Capabilities Registers Value..... 59

Figure 6.11. Device Control/Capabilities Register Bit Map..... 60

Figure 6.12. 32 DW Memory 32 Read Test Result 61

Figure 6.13. Complete Data Communication Cycle..... 62

LIST OF TABLES

Table 3.1. Transaction Types for Different Address Spaces 11

Table 3.2. Fmt[2:0] Field Values 15

Table 3.3. Fmt[2:0] and Type[4:0] Field Encodings 16

Table 3.4. Length[9:0] Field 16

Table 3.5. Address Field Bit Map 18

Table 3.6. Byte Enables Location and Correspondence 20

Table 3.7. Ordering Attributes 22

Table 3.8. No Snoop Attribute 22

Table 3.9. Definition of TC Field Encodings 23

Table 4.1. 7 Series FPGAs Integrated Block for PCI Express overview 24

CHAPTER ONE

Introduction

Background

Cancer is a major health threat. At the beginning of 2018, about 1.7 million people in the U.S. were expected to be diagnosed as new cancer cases and about 609,640 Americans were expected to die of cancer in 2017 [1]. Radiation therapy with protons and heavier ions is an attractive form of cancer treatment that could enhance local control and survival of cancers that are currently difficult to cure and lead to less side effects due to sparing of normal tissues [2]. Planning the energy and spatial distributions of the proton beam prior to treatment requires detailed knowledge of the "relative proton stopping power" (RSP) of the tissue in front of and in the tumor. X-ray CT scans are now used to estimate the RSP, but transforming from X-ray absorption to proton stopping power is ambiguous and error prone. Proton CT measures directly the RSP, and with minimal radiation dose (less than or no more than an X-ray CT scan) [3].

Single-particle tracking pCT technology has been put into clinical trials. Many recent publications reporting on pCT technology [4-10], pCT image quality [11,12], and mathematical and computer science aspects of pCT [13,14], demonstrating the large productivity in this field.

The existing Phase-II pCT scanner has been used in beam-test experiments not only by researchers but also by people not originally involved in its design and fabrication. Those experiments have generally required participation of an expert to

ensure success in setting up and operating the scanner. With some improvement and upgrade of hardware, we can simplify greatly the setup, calibration, and operation and make it possible for anybody in the research community to carry out an experiment with the device. At the same time, the scanner can run faster, cutting in half the time needed to make a full CT scan and reducing inefficiencies from pileup when using a pencil-beam, See Figure 1.1 for a photograph of pCT scanner.

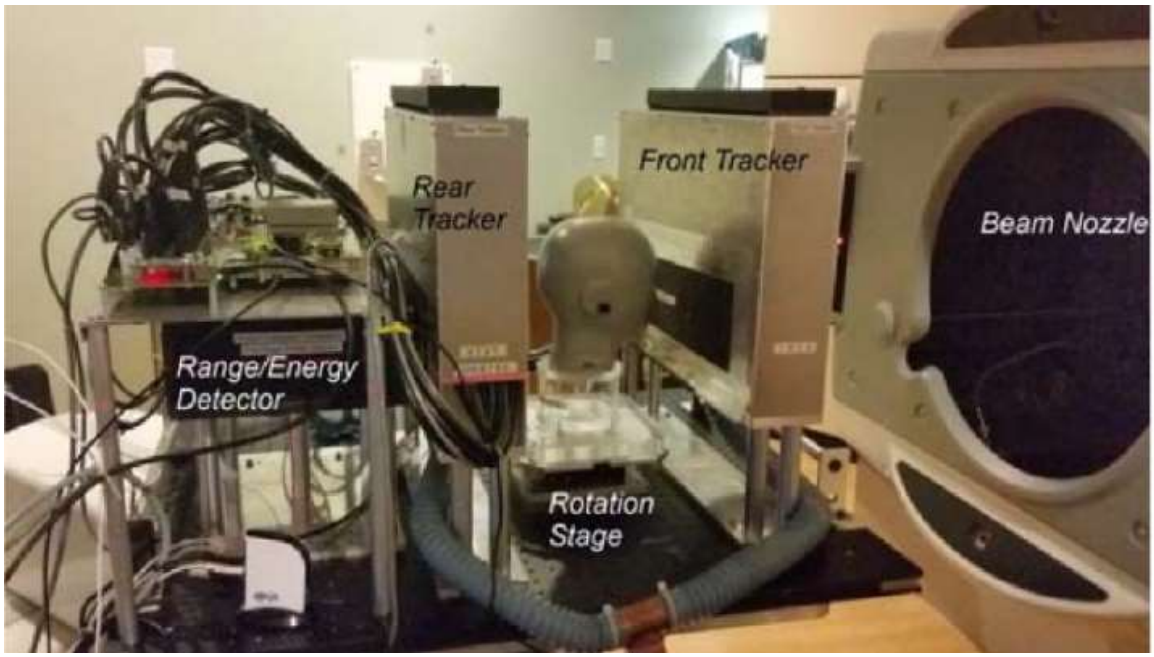


Figure 1.1. The Phase-II pCT scanner in the Northwestern Medicine Chicago Proton Center

The readout data of each energy detector and tracker is transmitted by a Spartan-6 FPGA to a Virtex-6 ‘event-builder’ Field Programmable Gate Array (FPGA) on the data acquisition board over 100 Mbit/s LVDS link. The ‘event-builder’ FPGA uses Ethernet for connection with Data Acquisition (DAQ) Computer. There are two key concerns of hardware upgrade:

1. Because the Ethernet link operates at 800 Mbit/s, the readout speed of events is limited at up to 1.2 MHz. Speed up the speed of link between ‘event-builder’ FPGA and DAQ Computer is a throttle of the performance of the current scanner.
2. The readout of data from the silicon-strip sensors is accomplished by a fully custom integrated circuit (ASIC) that was designed specifically for the Phase-II pCT scanner [15]. The redesign goal is to increase the speed of the preamplifier and shaping amplifiers by about a factor of four, such that the signal peaks in about 50 ns instead of 200 ns. This will greatly reduce the pileup probability, especially when running the system with a pencil beam, and can be accomplished by increasing the sizes of some of the transistors, especially the large input transistor, as well as the currents.

The thesis proposes a new interface design between ‘event-builder’ FPGA and DAQ based on PCI Express 3.0. By automated testing and results evaluation, the new design can speed up the original Ethernet link speed by a factor of 40, At the same time, supports the needs of the new signal peaks in 50 ns.

Organization of Thesis

Chapter two gives an overview of the PCI Express architecture. Some fundamental concepts of PCI Express are discussed, such as Link, Root Complex and Endpoint.

Chapter three discusses the behavior of the Transaction Layer of PCI Express.

Chapter four gives a brief introduction to Xilinx 7 Series FPGAs Integrated Block for PCI Express, including its transaction interface and flow control mechanism.

Chapter five discusses the design of user logic, including the Tx engine and the FWFT FIFO choice, and the Rx Engine.

Chapter six discusses the automated test of the design and test result, followed by the performance evaluation of the design.

Chapter seven makes a conclusion and gives future directions.

CHAPTER TWO

PCI Express Specification Introduction

This chapter gives an overview of the PCIE Express architecture and fundamental information to implement any PCI Express based logic design.

PCI Express Link

A link is a dual-simplex communication channel between two PCI Express components, such as a Root Complex to an Endpoint.

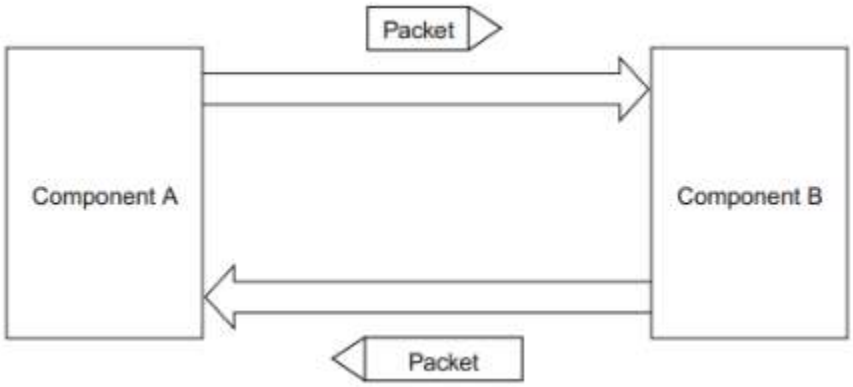


Figure 2.1. PCI Express Link

The basic PCI Express Link has two Low-Voltage Differential Signaling (LVDS) pairs: a Transmit Pair and a Receive Pair as shown in Figure 2.1. 8b/10b encoding is used to embed data clock (8b/10b is used in 5.0 Gigabits/second/lane, which this system uses, for 8.0 Gigabits/ second/ lane/ direction, 128b/130b encoding is used). Once a link is initialized, it only operates at one of the following speeds: for the first-generation PCI Express, only 2.5 Gigabits/second/lane/direction is supported; and for the second-

generation PCI Express, it supports an additional 5.0 Gigabits/second/lane/direction raw bandwidth (which this system uses); the third-generation adds an 8.0 Gigabits/second/lane/direction option.

A Link consists of at least one Lane, which is a set of LVDS pairs. For the scalation of bandwidth, a Link may use multiple Lanes, denoted by xN, where N is the Link widths (the third-generation PCI Express supports x1, x2, x4, x8, x12, x16, x32 while Xilinx 7 Series FPGAs Integrated Block for PCI Express supports only a subset of them). For example, an x8 Link of 5.0 GT/s data rate has an raw bandwidth of 40 Gigabits/second in each direction (which is also the case in this system).

After powering up, a PCI Express Link is set up following hardware initialization, when two components of a link negotiate lane widths and link speed. No software is involved in this process.

PCI Express Architecture

A PCI Express fabric is a set of Links that interconnect several components. A fabric example is shown in Figure 2.1, and consists of a Root Complex (RC) with main memory, PCI/PCI Express Bridge (optional), PCI Express Endpoints and/or Legacy Endpoints all connected by PCI Express Link.

Switch is not used in the system, and is thus not discussed here.

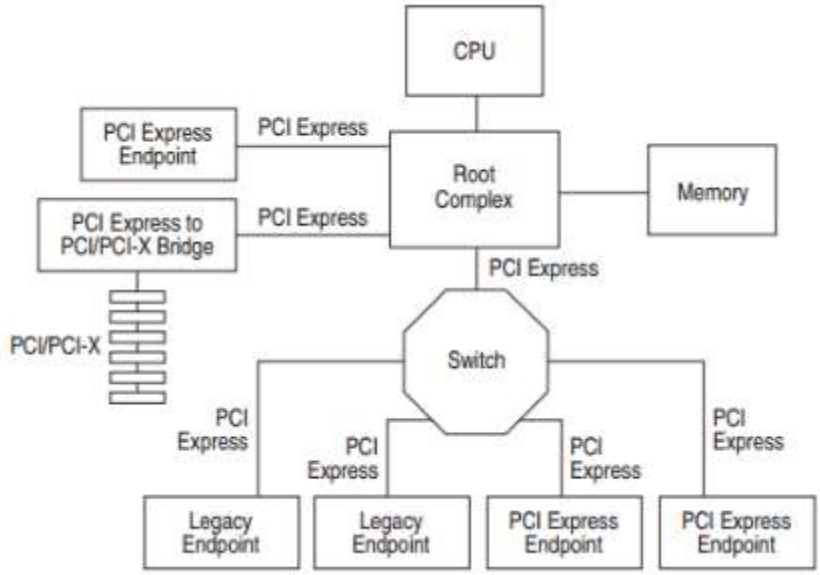


Figure 2.2. Fabric Example

Root Complex

A Root Complex (RC) connects the CPU/memory to other PCI Express Devices. A RC generates transactions (see chapter 3 for transaction types) on behalf of CPU, and processes the completed info when applied. Configuration requests must be supported by RC as a Requester. When the RC functions as a requester, it can choose to initialize an I/O Request but the choice is optional.

Endpoints

An endpoint can either be a Requester or Completer (used in this system). Endpoints types can be either legacy (not used in this system), PCI Express (used in this system), or Root Complex (used in this system, as a behavioral simulation model).

A Configuration Read Type 0 Transaction (with Type field in header of TLP set as 0) must be supported by the Endpoints as a Completer. One example of this is when the system initializes, the RC send configuration packets to get the information on

Endpoints, and maps its memory space. I/O Requests are not supported by PCI Express Endpoint.

PCI Express Layering

PCI Express has a three-layer structure, as shown in Figure 2.3. The Transaction Layer, the Data Link Layer, and the Physical Layer process outbound and inbound transactions coming from adjacent layers and present the results to next layers.

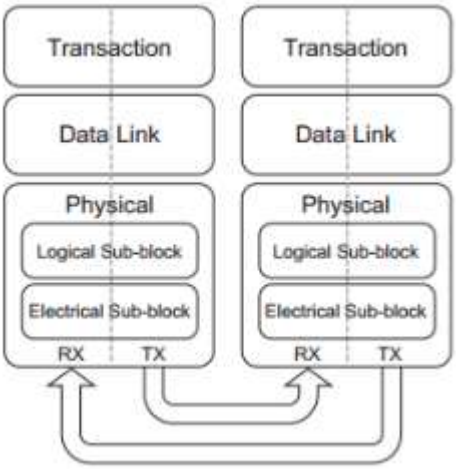


Figure 2.3. PCI Express Layering

Packets are used for communication between layers. Transaction Layer Packets (TLPs) are formed by the Transaction Layer and sent to the Data Link Layer to start a communication. As the packets are processed by the next 2 Layers, additional information is added to the head and tail parts of the packets. The additional information is used by the other side of the link, and so is removed from the layer which added it. The whole flow is shown in Figure 2.4.

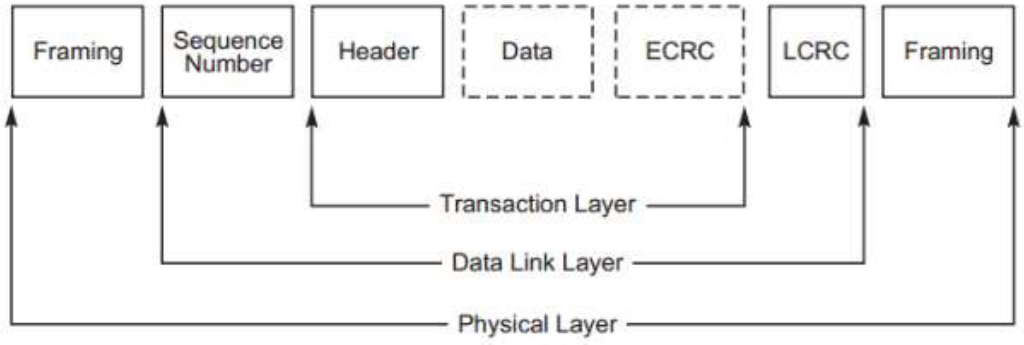


Figure 2.4. Packet Flow Through Layers

Transaction Layer

The top layer is the Transaction Layer. It is responsible for generating outbound TLPs and processing inbound TLPs. Credit-based flow control is performed at this layer. In this system, Xilinx 7 Series FPGAs Integrated Block for PCI Express IP gives the TLPs buffering/flow control information via the IP Core flow control interface. User design is responsible for the implementation.

The Transaction Layer supports four address spaces: memory, I/O, configuration (these are three implemented in the system) and Message.

Data Link Layer

The Data Link Layer is responsible for Link management and error detection/correction, serving between the Transaction Layer and the Physical Layer.

The transmission section of this Layer accepts TLPs generated by the Transaction Layer, extends them with data correction code and TLP sequence number, then transmits them to the Physical Layer. The receive section checks the data integrity of inbound TLPs, then transmits them to the Transaction Layer. When TLP errors happen, it is

responsible for generating retransmission request to the Transaction Layer, until the TLP is correctly processed, or the Link is determined to have failed.

Physical Layer

The lowest layer is the Physical Layer. It is responsible for input/output buffering, parallel-to-serial and serial-to-parallel conversion, 8b/10b or 128b/130b encoding/decoding, and impedance matching. It converts inbound packets from the Data Link Layer into a serialized format, and transmits it via physical medium (coaxial cords, fiber channel and so on) at a frequency and width negotiated during the initialization process.

The PCI Express architecture has “hooks” for future performance upgrading. The upgrade, if happens, can only affect the Physical Layer.

The logic design of this system is based on the 128-Bit Transaction Layer Interface provided by Xilinx 7 Series FPGAs Integrated Block for PCI Express IP (in the lowest level, using 7 Series FPGAs GTX/GTH Transceivers). Therefore, the Data Link Layer and the Physical Layer are not further discussed.

CHAPTER THREE

Transaction Layer Specification

As the top level, the Transaction Layer is responsible for:

1. Process of TLP in accordance with format transmit rules.
2. Management of Credit-based buffering/flow control.
3. Support of data poison and data integrity check (optional).
4. Support of the Virtual Transmit Channel (optional).

This chapter discusses the behavior of the Transaction Layer. Not all of the features of the Transaction Layer are covered; this chapter only focuses on the subset which is necessary for implementing a logic design working on the Transaction Layer. The implementation which involves many TLP format rules. For more details of The PCI Express Transaction Layer, see [16].

Transaction Types and Address Spaces

The two sides of Transaction are Requester and Completer. Four address spaces and associated Transaction types are defined, as shown in Table 3.1 for different usages.

Table 3.1. Transaction Types for Different Address Spaces.

Address Space	Transaction Types	Basic Usage
Memory	Read/Write	Transfer data to/from a memory-mapped location
I/O	Read/Write	Transfer data to/from a I/O-mapped location
Configuration	Read/Write	Device Function configuration/setup
Message	Baseline	From event signaling mechanism to general purpose to general purpose messaging

Note: Table 3.1. is referenced from [16].

Memory Transactions

Memory Transactions are the main components used for data transmission in this system. It includes:

1. Read Request/ Completion.
2. Write Request.
3. Atomic Operation Request/ Completion.

The first two types are used in this system.

Two address formats are used in Memory Transaction:

1. Short Address Format: 32-bit address.
2. Long Address Format: 64-bit address.

This system uses Short Address Format but Long Address Format is also supported.

I/O Transactions

I/O Transactions are used for PCI Express legacy devices. It may be deprecated by future versions of PCI Express. I/O Transactions include:

1. Read Request/ Completion
2. Write Request/ Completion
3. I/O Transactions only uses use a 32-bit Short Address Format.

The system doesn't use I/O Transactions.

Configuration Transactions

Configuration Transaction is used for the access of configuration registers of a target component. It is heavily used in software initialization and configuration process; its supported types include:

1. Read Request/ Completion
2. Write Request/ Completion

In this system, software initialization and configuration are finished by the Root Complex Simulation Model provided by Xilinx, to configure the Endpoint by operating its configuration registers.

Packet Format Overview

The Requester and Completer use packets to communicate with each other. The format of a Transaction Layer Packet (TLP) is shown in Figure 3.1 [16]. A TLP is divided into 4 parts based on the functionalities: TLP Prefixes (optional), TLP header, a data payload (when applicable), and an TLP digest(optional).

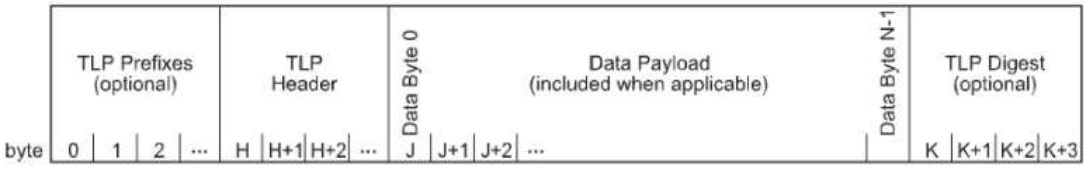


Figure 3.1. TLP Format

PCI Express Link transmits TLPs (with extended information added by Data Link Layer and Physical Layer) in a serialized form. At byte level, the leftmost byte is the first byte transmitted/received. For all the transactions generated by the user logic, Header is the first part, followed by Data Payload when applied. Note the byte order in PCI Express

SPEC differs from the order presented by Xilinx 7 Series FPGAs Integrated Block for PCI Express (see Chapter Four for more details).

TLP header consists of a subset of the following fields depending on its type:

1. Format of the packet
2. Type of the packet
3. Data Length
4. Transaction Descriptor (which consists of Transaction ID, Attributes, Traffic Class)
5. Address/ routing information
6. Byte Enables (1st BE and Last BE)
7. Message encoding
8. Completion status

Packet Definition

Transactions are performed by Requesters and Completers at the two sides of a link. The packet is the basic unit used by Transactions. Packet are classified by requests and completions. The request packet must be used by any Transaction, but completions are only used when they are applicable. For example, a read request requires returned data, and a I/O write request requires complete status information.

Packet Header

Packets may or may not have certain fields of the header, based on the transaction type, and some fields may have different bit lengths for different addressing formats. For a 32-bit-request TLP header, byte 4 is the requester ID field while for a completion TLP

header, it represents the completer ID. Byte 8-11 is address info for a request TLP, but it represents requester ID, tag and lower address filed for a completion TLP.

A typical TLP is shown in Figure 3.2. Because the logic design of this system should totally conform with PCI Express SPEC when generating outbound TLPs to Xilinx 7 Series FPGAs Integrated Block for PCI Express IP and processing inbound TLPs from it, the following sections discuss important subsets of [16].

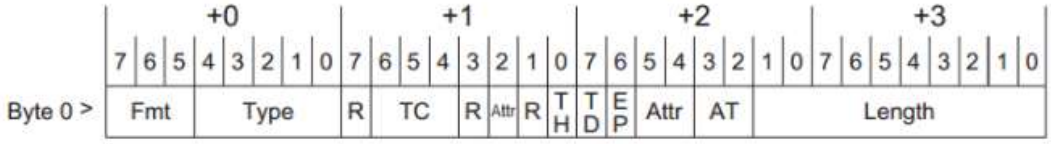


Figure 3.2. Byte 0-3 of TLP Header

The valid combination Fmt and Type fields determine the remaining parts of the TLP header, and whether or not data payload is following the header. All the valid values are shown in Table 3.2. and Table 3.3.

Table 3.2. Fmt[2:0] Field Values.

Fmt[2:0]	Corresponding TLP Format
3'b000	3 DW header, no data
3'b001	4 DW header, no data
3'b010	3 DW header, with data
3'b011	4 DW header, with data
3'b100	TLP Prefix
	All encodings not shown above are Reserved

Table 3.3. Fmt[2:0] and Type[4:0] Field Encodings.

TLP Type	Fmt[2:0]	Type[4:0]	Description
MRd	3'b000 3'b001	5'b00000	Memory Read Request
MRdLk	3'b000 3'b001	5'b00001	Memory Read Request-Locked
MWr	3'b010 3'b011	5'b00000	Memory Write Request
IORd	3'b000	5'b00010	I/O Read Request
IOWr	3'b010	5'b00010	I/O Write Request
CfgRd0	3'b000	5'b00100	Configuration Read Type 0
CfgWr0	3'b010	5'b00100	Configuration Write Type 0
CfgRd1	3'b000	5'b00101	Configuration Read Type 1
CfgWr1	3'b010	5'b00101	Configuration Write Type 1
TCfgRd	3'b000	5'b11011	Deprecated TLP Type
TCfgWr	3'b010	5'b11011	Deprecated TLP Type
Msg	3'b001	5'b10$r_2r_1r_0$	Message Request – The sub-field r[2:0] specifies the Message routing mechanism.
MsgD	3'b011	5'b10$r_2r_1r_0$	Message Request with data payload – The sub-field r[2:0] specifies the Message routing mechanism.
Cpl	3'b000	5'b01010	Completion without Data – Used for I/O and Configuration Write Completions with any Completion Status. Also used for Atomic Operation Completions and Read Completions (I/O, Configuration, or Memory) with Completion Status other than Successful Completion.
CplD	3'b010	5'b01010	Completion with Data – Used for Memory, I/O and Configuration Read Completions. Also used for Atomic Operation Completions.
CplLk	3'b000	5'b01011	Completion for Locked Memory Read without Data = Used only in error case.
CplDLk	3'b010	5'b01011	Completion for Locked Memory Read – otherwise lick CplD.

Length field determines the DW (four bytes) length of the data payload, if presented. Its valid values are shown in Table 3.4.

Table 3.4. Length[9:0] Field.

Length[9:0]	Corresponding TLP Data Payload Size
9'b000000000	1024 DW
9'b000000001	1 DW
…	…
9'b111111111	1023 DW

Note that 9'b000000000 represents maximum 1024 DW allowed by PCI Express since all the transaction must not cross 4 KB boundary).

A TLP with a data payload must limit the payload size within the minimum value denoted by Max_Payload_Size of the Device Control Registers of Requester and Completer of Link. This system supports a Max_Payload_Size of 512 KB in real application, while the simulation and evaluation only use 128 KB of it because the other side of the Link, Root Complex simulation model only supports a Max_Payload_Size of 128 KB. TLPs violating this rule are Malformed TLPs and discarded.

Note the size of a Memory Request Size is also limited by Max_Read_Request_Size of the Device Control Registers of the devices. Also note that the rule doesn't count TLP Digest as Payload Length; only data is counted.

Routing of TLP

Address, ID and implicit routing are defined in the Express SPEC. Currently, only Address Based routing is used in this system because memory r/w is the main transaction used in this system. The other 2 routing mechanisms are not discussed here.

Address routing is used in Memory and I/O Requests. As shown in Figure 3.3. and Figure 3.4., it supports 12-byte (3 DW) header with 64-bit address format, and 16-byte (4 DW) with 32-bit address format.

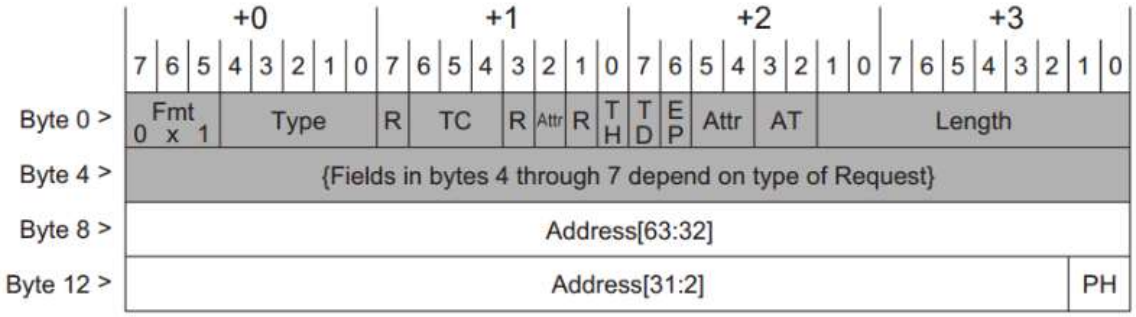


Figure 3.3. 64-bit Format Routing

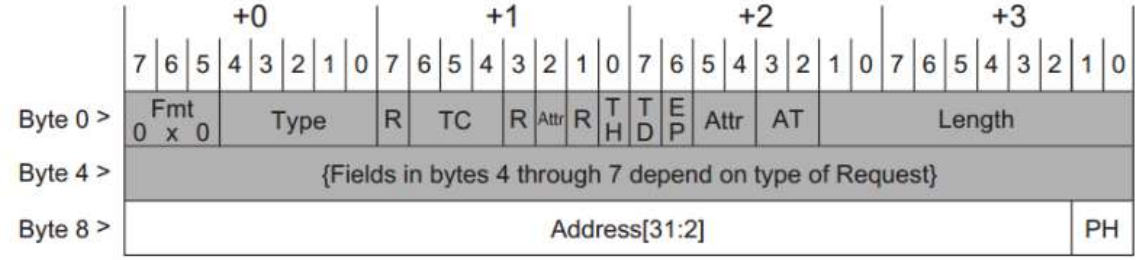


Figure 3.4. 32-bit Format Routing

Address field bit map is given by Table 3.5.

Table 3.5. Address Field Bit Map

Address Bits	32-bit Addressing	64-bit Addressing
63:56	Not Applicable	Bits 7:0 of Byte 8
55.48	Not Applicable	Bits 7:0 of Byte 9
47:40	Not Applicable	Bits 7:0 of Byte 10
39:32	Not Applicable	Bits 7:2 of Byte 11
31:24	Bits 7:0 of Byte 8	Bits 7:0 of Byte 12
23:16	Bits 7:0 of Byte 9	Bits 7:0 of Byte 13
15:8	Bits 7:0 of Byte 10	Bits 7:0 of Byte 14
7:2	Bits 7:2 of Byte 11	Bits 7:2 of Byte 15

Memory Read/Write, and Atomic Operation Requests can use either 32-bit address format or 64-bit address format. 32-bit format must be used for Addresses smaller than 4 GB. A use of 64-bit address format for addresses smaller than 4 GB is

undefined in PCI Express SPEC. In this system, all the TLPs use 32-bit format because the mapped address is in Mega Bytes level.

Byte Enables Rules

1st DW BE and Last DW BE are used in all the transaction types except Message. They are at the seventh byte of the header, as shown in Figure 3.5.

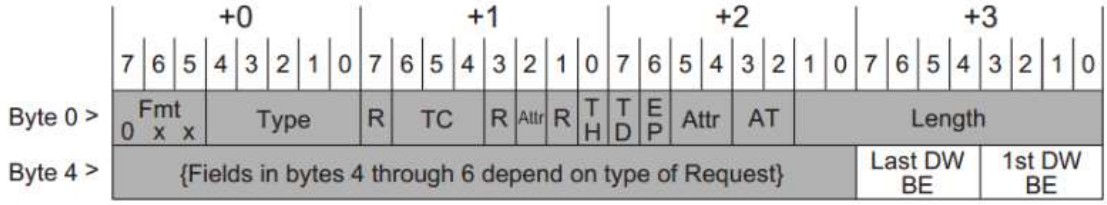


Figure 3.5. 1st /Last DW BEs

1st DW BE and Last DW BE both have 4 bits, with each bit indicating the byte enable for corresponding byte location in 1st/Last DW of the TLP payload.

If a request TLP has a Length field larger than 1 DW, which is 4 bytes, then 1st DW BE should not be 0000b and Last DW BE should not be 000b. If a request TLP has a Length field of 1 DW, Last DW BE can only be 0000b.

For all 2 DW Memory Type Requests that are not QW aligned, and Memory Type Requests that are larger or equal to 3 DW, the 1st DW BE and Last DW BE fields must be set to a contiguous format, for example:

2 DW Length Field, 1st DW BE 1000b, Last DW BE 0111b.

Table 3.6. shows the bit map for 1st /Last Byte Enable fields,

Table 3.6. Byte Enables Location and Correspondence

Byte Enables	Header Location	Affected Data Byte
1st DW BE[0]	Bits 0 of Byte 7	Byte 0
1st DW BE[1]	Bits 1 of Byte 7	Byte 1
1st DW BE[2]	Bits 2 of Byte 7	Byte 2
1st DW BE[3]	Bits 3 of Byte 7	Byte 3
Last DW BE[0]	Bits 4 of Byte 7	Byte N-4
Last DW BE[1]	Bits 5 of Byte 7	Byte N-3
Last DW BE[2]	Bits 6 of Byte 7	Byte N-2
Last DW BE[3]	Bits 7 of Byte 7	Byte N-1

Note: Referenced from [16]

Note 1 DW Read Request with 1st DW BE set to 0000b is permitted. In this case, the Completion-with-Data TLP must include 1 DW data payload with any value.

Transaction Descriptor

The Transaction Descriptor enables the Identification between different Devices and is the fundamental functionality for PCI Express Packets Ordering. As shown in Figure 3.6., it consists of Transaction ID, Attributes and Traffic Class (TC). Note these fields are not in a contiguous position within the packet header.

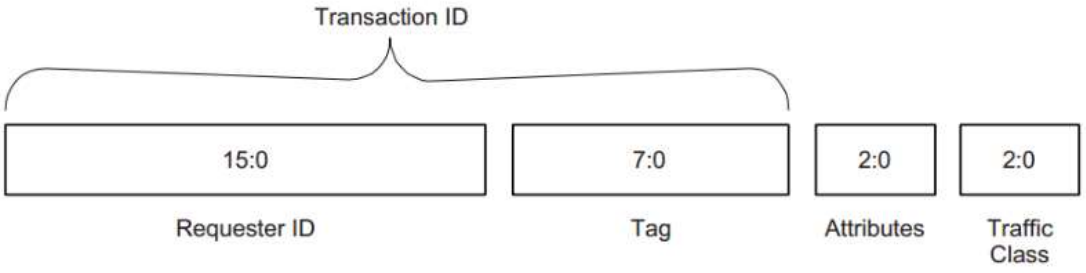


Figure 3.6. Transaction Descriptor

Transaction ID. As shown in Figure 3.7., Transaction ID has two parts: Requester ID and Tag as shown in Figure 3.7. The requester ID is a 16-bit unique value for every device in a PCI Express Fabric. The tag is generated and maintained

independently by every device, and it is unique for every outstanding TLP generated by that device. With the Transaction ID, any request TLP in a PCI Express Fabric can be identified.

Figure 3.7. Transaction ID

Attributes. The Attributes field is a 3-bit field used for TLP Ordering and Hardware coherency management (snoop), as shown in Figure 3.8.

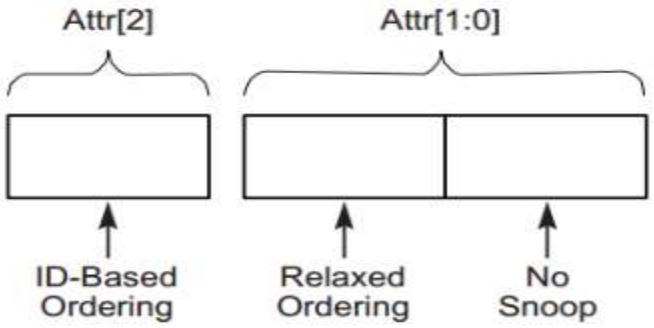


Figure 3.8. Attributes Field

Table 3.7. defines the states of the Relaxed Ordering and ID-Based Ordering attribute fields. The system uses the PCI Strongly Ordered Model because it has only one Root Complex (Requester) and one Endpoint (Completer). So all the TLPs transmitted in this system has the field Attr[2:1] as 2'b00.

Table 3.7. Ordering Attributes

Attr[2]	Attr[1]	Cache Coherency Management Type	Coherency Model
0	0	Default Ordering	PCI Strongly Ordered Model
0	1	Relaxed Ordering	PCI-X Relaxed Ordering Model
1	0	ID-Based Ordering	Independent ordering based on Requester/Completer ID
1	1	Relaxed Ordering plus ID-Based Ordering	Logical "OR" Relaxed Ordering and ID-Based Ordering

Note: Referenced from [16]

No Snoop Attribute. The Table 3.8. shows the definition of the No Snoop attribute.

Table 3.8. No Snoop Attribute

No Snoop Attribute	Cache Coherency Management Type	Coherency Model
0	Default	Hardware enforced cache coherency expected
1	No Snoop	Hardware enforced cache coherency not expected

Note: Referenced from [16]

For all the Transaction Types used in this system, the hardware enforced cache coherency model is used.

Traffic Class (TC). TC is a 3-bit field used for supporting the PCI Express Virtual Channel. It is the fundamental functionality for the PCI Express Bus Arbitration Process. In this system, all the TLPs have TC field set as 3'b000. Table 3.9. defines the TC encodings.

Table 3.9. Definition of TC Field Encodings

TC Field Value	Definition
000	TC0: Best Effort service class (General Purpose I/O), Default TC – must be supported by every PCI Express device
001 - 111	TC1 -TC7: Differentiated service classes (Differentiation based on Weighted-Round-Robin and/or Priority)

Note: Referenced from [16]

CHAPTER FOUR

Introduction to Xilinx 7 Series FPGAs Integrated Block for PCI Express

In this system, the user design works with Xilinx 7 Series FPGAs Integrated Block for PCI Express to provide Endpoint function. Therefore, before implementing any features in and above the Transaction Layer, the fundamental information about Endpoint IP must be discussed, including the Transaction Layer AXI4-Steam Interface and Core Buffering/Flow Control mechanism. For more details about Xilinx 7 Series FPGAs Integrated Block for PCI Express, please check Xilinx 7 Series FPGAs Integrated Block for PCI Express LogiCORE IP Product Guide [17].

Endpoints support 2.5 Gb/s and 5.0 Gb/s lane speeds, with different AXI4 data bus width and lane widths, as shown in Table 4.1.

Table 4.1. 7 Series FPGAs Integrated Block for PCI Express overview

Name	User Interface	Supported Lane Widths
1-lane at 2.5 Gb/s, 5.0 Gb/s	64	x1
2-lane at 2.5 Gb/s, 5.0 Gb/s	64	x1, x2
4-lane at 2.5 Gb/s, 5.0 Gb/s	64, 128	x1, x2, x4
8-lane at 2.5 Gb/s, 5.0 Gb/s	64, 128	x1, x2, x4, x8

Note: Referenced from [17]

The Xilinx 7 Series FPGAs Integrated Block for PCI Express core provides full functionality in Transaction Layer, Data Link and Physical Layer, conforming to the PCI Express Base Specification.

Figure 4.1 shows the structure of the 7 Series FPGAs Integrated Block for PCI Express IP core. In this system, the IP is configured to x8 Link of 5.0 GT/s, 128-bit wide AXI4 interface mode.

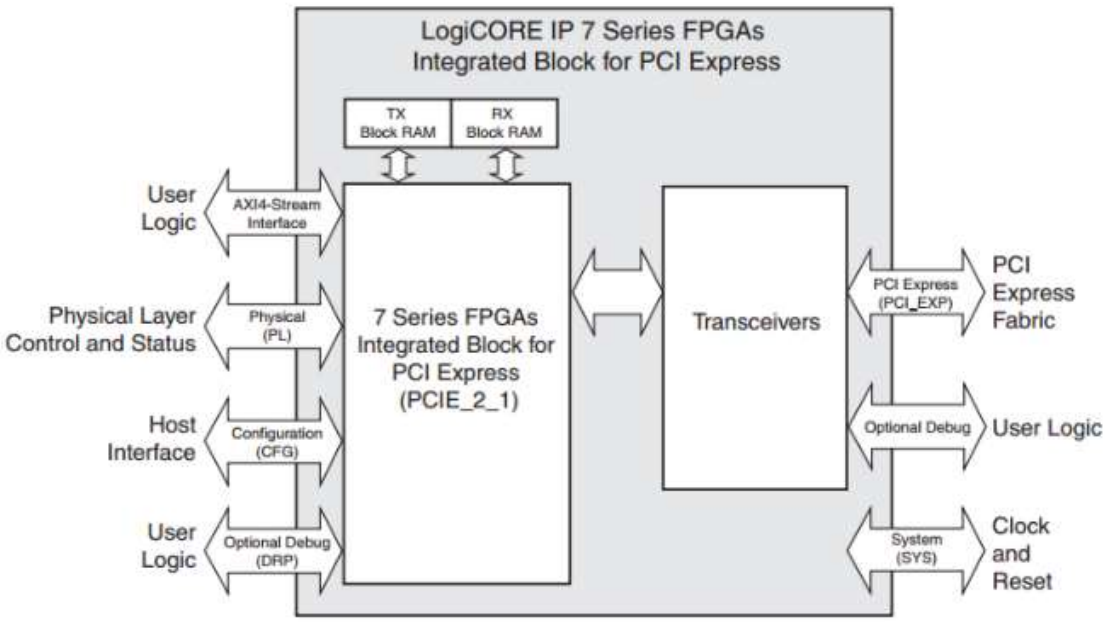


Figure 4.1. Top Level Functional Blocks and Interfaces [17]

Transaction Interface

For outbound transmission, TLPs are generated by user logic and sent to the Transaction Interface. For inbound transmission, TLPs are presented by Endpoint IP and consumed by user logic. Since user logic focuses on the Transmit Interface, Receive Interface and Configuration Interface, other core interfaces are not discussed.

Figure 4.2., Figure 4.3. and Figure 4.4. show the definition of signals of Transmit Interface and Receive Interface [17].

<table>
<tr><th>Name</th><th>Mnemonic</th><th>Direction</th><th>Description</th></tr>
<tr><td>s_axis_tx_tlast</td><td></td><td>Input</td><td>Transmit End-of-Frame (EOF): Signals the end of a packet. Valid only along with assertion of s_axis_tx_tvalid.</td></tr>
<tr><td>s_axis_tx_tdata[W-1:0]</td><td></td><td>Input</td><td>Transmit Data: Packet data to be transmitted.
<table><tr><th>Product</th><th>Data Bus Width (W)</th></tr><tr><td>1-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64</td></tr><tr><td>2-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64</td></tr><tr><td>4-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64 or 128</td></tr><tr><td>8-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64 or 128</td></tr></table></td></tr>
<tr><td>s_axis_tx_tkeep[7:0] (64-bit interface)

s_axis_tx_tkeep[15:0] (128-bit interface)</td><td></td><td>Input</td><td>Transmit Data Strobe: Determines which data bytes are valid on s_axis_tx_tdata[W-1:0] during a given beat (s_axis_tx_tvalid and s_axis_tx_tready both asserted).
• Bit 0 corresponds to the least significant byte on s_axis_tx_tdata.
• Bit 7 (64-bit) and bit 15 (128-bit) correspond to the most significant byte.
For example:
• s_axis_tx_tkeep[0] ==1b, s_axis_tx_tdata[7:0] is valid
• s_axis_tx_tkeep[7] ==0b, s_axis_tx_tdata[63:56] is not valid
When s_axis_tx_tlast is not asserted, the only valid values are 0xFF (64-bit) or 0xFFFF (128-bit).
When s_axis_tx_tlast is asserted, valid values are:
• 64-bit: only 0x0F and 0xFF are valid.
• 128-bit: 0x000F, 0x00FF, 0x0FFF, and 0xFFFF are valid.</td></tr>
<tr><td>s_axis_tx_tvalid</td><td></td><td>Input</td><td>Transmit Source Ready: Indicates that the user application is presenting valid data on s_axis_tx_tdata.</td></tr>
<tr><td>s_axis_tx_tready</td><td></td><td>Output</td><td>Transmit Destination Ready: Indicates that the core is ready to accept data on s_axis_tx_tdata. The simultaneous assertion of s_axis_tx_tvalid and s_axis_tx_tready marks the successful transfer of one data beat on s_axis_tx_tdata.</td></tr>
<tr><td>s_axis_tx_tuser[3]</td><td>t_src_dsc</td><td>Input</td><td>Transmit Source Discontinue: Can be asserted any time starting on the first cycle after start-of-frame (SOF). Assert s_axis_tx_tlast simultaneously with (tx_src_dsc)s_axis_tx_tuser[3].</td></tr>
<tr><td>tx_buf_av[5:0]</td><td></td><td>Output</td><td>Transmit Buffers Available: Indicates the number of free transmit buffers available in the core. Each free transmit buffer can accommodate one TLP up to the supported maximum payload size (MPS). The maximum number of transmit buffers is determined by the supported MPS and block RAM configuration selected. (See Core Buffering and Flow Control, page 93.)</td></tr>
</table>

Figure 4.2. Transmit Interface

<table>
<tr><th>Name</th><th>Mnemonic</th><th>Direction</th><th>Description</th></tr>
<tr><td>s_axis_tx_tlast</td><td></td><td>Input</td><td>Transmit End-of-Frame (EOF): Signals the end of a packet. Valid only along with assertion of s_axis_tx_tvalid.</td></tr>
<tr><td>s_axis_tx_tdata[W-1:0]</td><td></td><td>Input</td><td>Transmit Data: Packet data to be transmitted.
<table><tr><th>Product</th><th>Data Bus Width (W)</th></tr><tr><td>1-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64</td></tr><tr><td>2-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64</td></tr><tr><td>4-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64 or 128</td></tr><tr><td>8-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64 or 128</td></tr></table></td></tr>
<tr><td>s_axis_tx_tkeep[7:0] (64-bit interface)

s_axis_tx_tkeep[15:0] (128-bit interface)</td><td></td><td>Input</td><td>Transmit Data Strobe: Determines which data bytes are valid on s_axis_tx_tdata[W-1:0] during a given beat (s_axis_tx_tvalid and s_axis_tx_tready both asserted).
• Bit 0 corresponds to the least significant byte on s_axis_tx_tdata.
• Bit 7 (64-bit) and bit 15 (128-bit) correspond to the most significant byte.
For example:
• s_axis_tx_tkeep[0] ==1b, s_axis_tx_tdata[7:0] is valid
• s_axis_tx_tkeep[7] ==0b, s_axis_tx_tdata[63:56] is not valid
When s_axis_tx_tlast is not asserted, the only valid values are 0xFF (64-bit) or 0xFFFF (128-bit).
When s_axis_tx_tlast is asserted, valid values are:
• 64-bit: only 0x0F and 0xFF are valid.
• 128-bit: 0x000F, 0x00FF, 0x0FFF, and 0xFFFF are valid.</td></tr>
<tr><td>s_axis_tx_tvalid</td><td></td><td>Input</td><td>Transmit Source Ready: Indicates that the user application is presenting valid data on s_axis_tx_tdata.</td></tr>
<tr><td>s_axis_tx_tready</td><td></td><td>Output</td><td>Transmit Destination Ready: Indicates that the core is ready to accept data on s_axis_tx_tdata. The simultaneous assertion of s_axis_tx_tvalid and s_axis_tx_tready marks the successful transfer of one data beat on s_axis_tx_tdata.</td></tr>
<tr><td>s_axis_tx_tuser[3]</td><td>t_src_dsc</td><td>Input</td><td>Transmit Source Discontinue: Can be asserted any time starting on the first cycle after start-of-frame (SOF). Assert s_axis_tx_tlast simultaneously with (tx_src_dsc)s_axis_tx_tuser[3].</td></tr>
<tr><td>tx_buf_av[5:0]</td><td></td><td>Output</td><td>Transmit Buffers Available: Indicates the number of free transmit buffers available in the core. Each free transmit buffer can accommodate one TLP up to the supported maximum payload size (MPS). The maximum number of transmit buffers is determined by the supported MPS and block RAM configuration selected. (See Core Buffering and Flow Control, page 93.)</td></tr>
</table>

Figure 4.2. Transmit Interface (Cont'd)

<table>
<tr><th>Name</th><th>Mnemonic</th><th>Direction</th><th>Description</th></tr>
<tr><td>m_axis_rx_tlast</td><td></td><td>Output</td><td>Receive End-of-Frame (EOF):
Signals the end of a packet. Valid only if m_axis_rx_tvalid is also asserted.
The 128-bit interface does not use m_axis_rx_tlast signal at all (tied Low), but rather it uses m_axis_rx_tuser signals.</td></tr>
<tr><td>m_axis_rx_tdata[W-1:0]</td><td></td><td>Output</td><td>Receive Data:
Packet data being received. Valid only if m_axis_rx_tvalid is also asserted.
When a Legacy Interrupt is sent from the Endpoint, the ENABLE_MSG_ROUTE attribute should be set to 11'b00000001000 to see this signal toggling along with m_axis_rx_tdata, m_axis_rx_tkeep and m_axis_rx_tuser.
<table><tr><th>Product</th><th>Data Bus Width (W)</th></tr><tr><td>1-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64</td></tr><tr><td>2-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64</td></tr><tr><td>4-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64 or 128</td></tr><tr><td>8-lane (2.5 Gb/s and 5.0 Gb/s)</td><td>64 or 128</td></tr></table>128-bit interface only: Unlike the Transmit interface s_axis_tx_tdata[127:0], received TLPs can begin on either the upper QWORD m_axis_rx_tdata[127:64] or lower QWORD m_axis_rx_tdata[63:0] of the bus. See the description of is_sof and (rx_is_sof[4:0]) m_axis_rx_tuser[14:10] m_axis_rx_tuser[21:17] for further explanation.</td></tr>
<tr><td>m_axis_rx_tuser[14:10]
(128-bit interface only)</td><td>rx_is_sof[4:0]</td><td>Output</td><td>Indicates the start of a new packet header in m_axis_rx_tdata:
• Bit 4: Asserted when a new packet is present
• Bit 0-3: Indicates byte location of start of new packet, binary encoded
Valid values:
• 5'b10000 = SOF at AXI byte 0 (DWORD 0) m_axis_rx_tdata[7:0]
• 5'b11000 = SOF at AXI byte 8 (DWORD 2) m_axis_rx_tdata[71:64]
• 5'b00000 = No SOF present</td></tr>
</table>

Figure 4.3. Receive Interface

Name	Mnemonic	Direction	Description
m_axis_rx_tuser[21:17] (128-bit interface only)	rx_is_eof[4:0]	Output	Indicates the end of a packet in m_axis_rx_tdata: • Bit 4: Asserted when a packet is ending • Bit 0-3: Indicates byte location of end of the packet, binary encoded Valid values: • 5'b10011 = EOF at AXI byte 3 (DWORD 0) m_axis_rx_tdata[31:24] • 5'b10111 = EOF at AXI byte 7 (DWORD 1) m_axis_rx_tdata[63:56] • 5'b11011 = EOF at AXI byte 11 (DWORD 2) m_axis_rx_tdata[95:88] • 5'b11111 = EOF at AXI byte 15 (DWORD 3) m_axis_rx_tdata[127:120] • 5'b01111 = No EOF present
m_axis_rx_tuser[1]	rx_err_fwd	Output	Receive Error Forward: • *64-bit interface:* When asserted, marks the packet in progress as error-poisoned. Asserted by the core for the entire length of the packet. • *128-bit interface:* When asserted, marks the current packet in progress as error-poisoned. Asserted by the core for the entire length of the packet. If asserted during a straddled data transfer, applies to the packet that is beginning.
m_axis_rx_tuser[0]	rx_ecrc_err	Output	Receive ECRC Error: Indicates the current packet has an ECRC error. Asserted at the packet EOF.
m_axis_rx_tvalid		Output	Receive Source Ready: Indicates that the core is presenting valid data on m_axis_rx_tdata.
m_axis_rx_tready		Input	Receive Destination Ready: Indicates that the user application is ready to accept data on m_axis_rx_tdata. The simultaneous assertion of m_axis_rx_tvalid and m_axis_rx_tready marks the successful transfer of one data beat on s_axis_tx_tdata. For a Root port configuration, when a Legacy Interrupt is sent from the Endpoint, the ENABLE_MSG_ROUTE attribute should be set to 11'b00000001000 to see this signal toggling along with m_axis_rx_tdata, m_axis_rx_tkeep and m_axis_rx_tuser.
m_axis_rx_tuser[9:2]	rx_bar_hit[7:0]	Output	Receive BAR Hit: Indicates BAR(s) targeted by the current receive transaction. Asserted from the beginning of the packet to m_axis_rx_tlast. • (rx_bar_hit[0])m_axis_rx_tuser[2]: BAR0 • (rx_bar_hit[1])m_axis_rx_tuser[3]: BAR1 • (rx_bar_hit[2])m_axis_rx_tuser[4]: BAR2 • (rx_bar_hit[3])m_axis_rx_tuser[5]: BAR3 • (rx_bar_hit[4])m_axis_rx_tuser[6]: BAR4 • (rx_bar_hit[5])m_axis_rx_tuser[7]: BAR5

Figure 4.3. Receive Interface (Cont'd)

Figure 4.4. shows a subset of Configuration Interface Signals which are important for user logic [17].

Name	Direction	Description
cfg_status[15:0]	Output	Configuration Status: Status register from the Configuration Space Header. Not supported.
cfg_command[15:0]	Output	Configuration Command: Command register from the Configuration Space Header.
cfg_dstatus[15:0]	Output	Configuration Device Status: Device status register from the PCI Express Capability Structure.
cfg_dcommand[15:0]	Output	Configuration Device Command: Device control register from the PCI Express Capability Structure.

Figure 4.4. A Subset of Configuration Interface

AXI4 stream data bus has a reverse endianness with PCI Express. Figure 4.5. represents a typical 32-bit addressable Memory TLP on the AXI4 data bus [17].

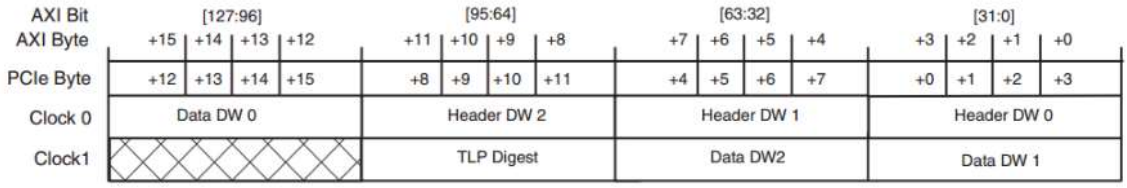


Figure 4.5. Memory 32 TLP on AXI4 Interface

Both 3 DW and 4 DW header TLPs are supported by AXI4 interface.

Figure 4.6. shows the typical timing of a 3 DW header TLP with 8 DW payload sent by user logic. User logic asserts the tx_valid signal, and at the same time presents the 3 DW header with 1 DW payload on AXI4 data bus. tkeep[15:0] is set to ffffh to notify the Endpoint IP that all 4 DW at this cycle contain valid data. At the last cycle, sser logic asserts tx_last signal to notify Endpoint IP that last frame of data is presented, with tkeep[15:0] set to 0fffh, notifying the Endpoint IP that only lower 3 DW is valid payload.

Figure 4.7. represents a 4 DW Memory 64 TLP with n DW data payload, note that tkeep[15:0] is set to ffffh except in the last cycle, only the lower 2 DW is valid payload.

Receive Interface has a similar timing with the only difference being that Endpoint IP is the master side.

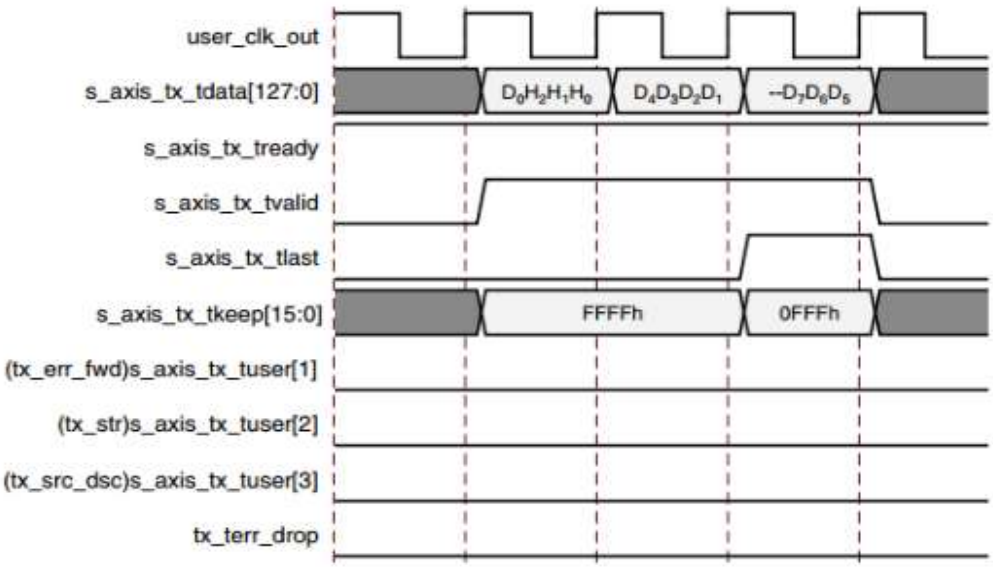


Figure 4.6. 3 DW TLP with Payload

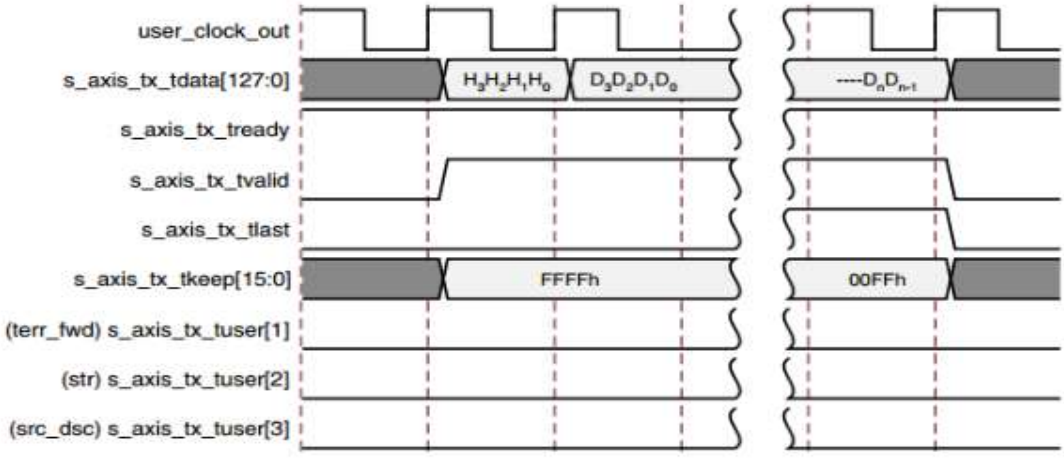


Figure 4.7. 4 DW Header with Payload

Credit-Based Core Buffering/Flow Control

In the initialization process, Root Complex configures the Device Control Register of Endpoint, which includes a MAX_PAYLOAD_SIZE field. The maximum size of TLP payload is limited by the smaller MAX_PAYLOAD_SIZE of both the RC and the Endpoint.

PCI Express uses a Credit-Based Flow Control mechanism. Requester and Completer have their own buffering space. For every outstanding request or completion TLP, one credit is consumed which indicates that one buffering space is used. When the RC is receiving the corresponding Completion TLP or Link partner successfully receives a TLP, and one credit is restored.

In this design, the Xilinx 7 Series FPGAs Integrated Block for PCI Express core have 32 credit/buffering space, represented by tx_buf_av signal on transmit interface, with each storing a TLP with MAX_PAYLOAD_SIZE. A TLP is backed up in a buffering space until it is successfully received by other side of the link.

Any buffering space can hold only one TLP at any time, no matter the size of the TLP. An abnormal flow control behavior of the RC is discussed in Chapter Six. More details can be found in [17].

CHAPTER FIVE

A Transmit-Receive Engine Based Logic Design

This chapter discusses a Transmit-Receive Engine Based Logic Design. The design hierarchy is shown in Figure 5.1. The design is based on the PIO design provided by XILINX, with modification and custom features which are proved to be able to satisfy two key concerns of the Phase-II pCT scanner hardware upgrade:

1. It increases the link speed from 800 Mbit/s to 26.491 Gbit/s, by about 33 times.
2. The logic is able to handle signal peaks in 50-ns level rather than 200-ns level in the existing Phase-II pCT scanner.

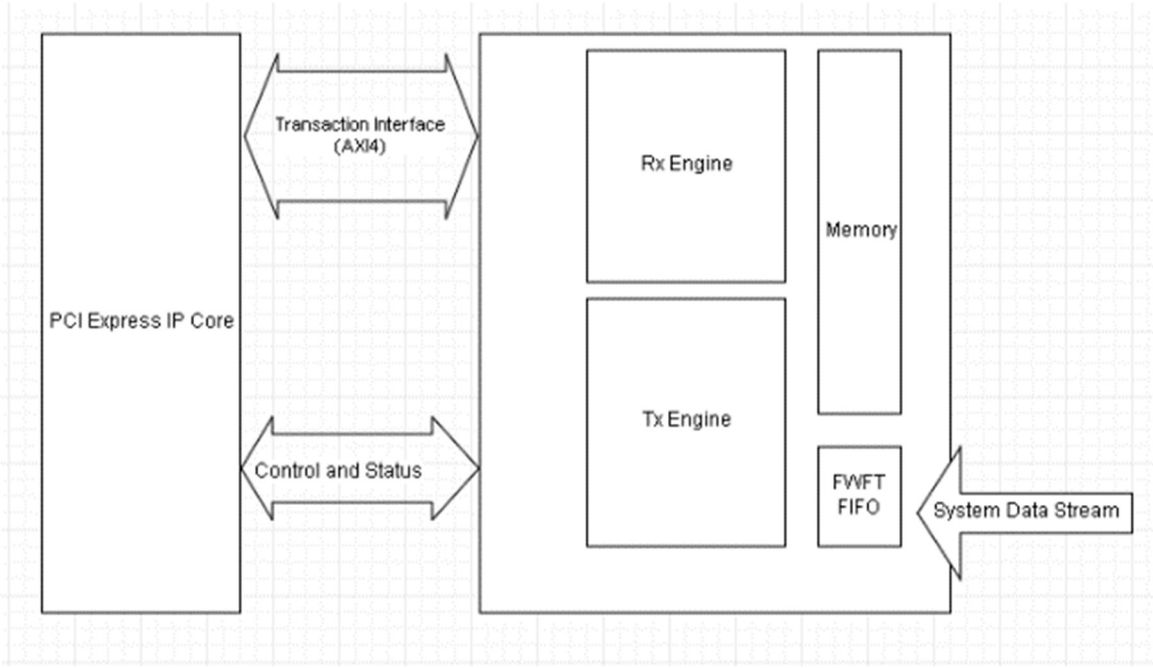


Figure 5.1. Top Level Hierarchy

This design uses BRAM as memory mapped address space for the Endpoint, also it uses a first-word-fall-through FIFO to buffer the System Data Stream. Memory Read 32 TLP and Memory Write 32 TLP are the main transactions. Memory Read 64, Memory Write 64 and I/O Read/Write are also supported.

When an inbound TLP is received, the design checks the TLPs that target destination and TLP header info. If the TLP hits the BAR space and the TLP header info conforms to the rules discussed in Chapter Two, then the Rx Engine processes the TLP. If not, the Rx Engine informs the TX Engine that a TLP is invalid, and the TX Engine provides the error info to PCI Express IP Core.

For a valid TLP, if it is a Memory Write 32 TLP, the Rx Engine extracts the header info, processes it, and writes the data offset to the corresponding address of the mapped memory. No completion TLP is given back in this case.

For a Memory Read 32 TLP, the Rx Engine extracts header info such as address, Transaction Descriptor, Request Length, Byte Enables and passes them to the Tx Engine. Based on the header info, the Tx Engine generates correct completion with data TLP, and reads data either from mapped memory or FWFT FIFO. After the Tx Engine successfully sends the completion with data TLP to the IP Core, it also generates a request complete signal for the Rx engine.

Based on the flow control signal given by the IP Core, the Tx Engine may throttle the data stream by de-asserting axi_data_valid signal and stop reading data from FIFO. If the credits and buffer space of the Endpoint is not enough, Rx and Tx engine will suspend all the ongoing transmission, and wait for new credit and buffer space to become available.

Since I/O Read/Write is not the main transaction types used in the system, the processing of them will not be further discussed here. But note that for an inbound I/O Write TLP, the Tx Engine also generates a completion without data TLP, according to the transaction rules of PCI Express SPEC.

Memory 64 TLPs are handled in a similar way with Memory 32 TLPs but with different TLP format structure. Because the system doesn't require a mapped endpoint larger than 4 GB, Memory 64 TLPs are not used.

Tx Engine and FWFT FIFO Choice

The state diagram of Tx Engine is shown in Figure 5.2. After detecting a request from Rx Engine, it first check if all the rules of the transaction (see Chapter Two or PCI Express SPEC for more details) are satisfied. If any error is detected, it reports the error to Xilinx 7 Series FPGAs Integrated Block for PCI Express. Either an error drop in the buffer pool of IP core or a retransmit happens in this case.

Because the system uses Memory 32 TLP as the main transaction type, only Memory 32 TLPs related states are discussed here. If the TLP is a Memory 32 read, the Tx Engine generates a 3 DW TLP header and tries to present it on 2 available cycles of the IP core. If the destination throttle happens (for example IP de-asserts axi_ready signal), then the Tx engine waits for its available cycle.

After successfully sending the header, the Tx engine fetches data from BRAM (for normal destination address)/ FIFO (for specific address mapped to transmitting Phase-II pCT scanner detector's stream data) at a speed of 128 bit/cycle, and sends the fetched data to the IP's Transaction Layer interface. The same process is performed when a destination throttle happens. When the Tx engine is sending the last 4 DW data (128

bits), it also asserts axi_last signal to notify the IP that all the data payload is sent. Note that depending on the destination address and requested length, the Tx engine also needs to operate on axi_keep[15:0] signal and set the corresponding Byte Enables fields in the header.

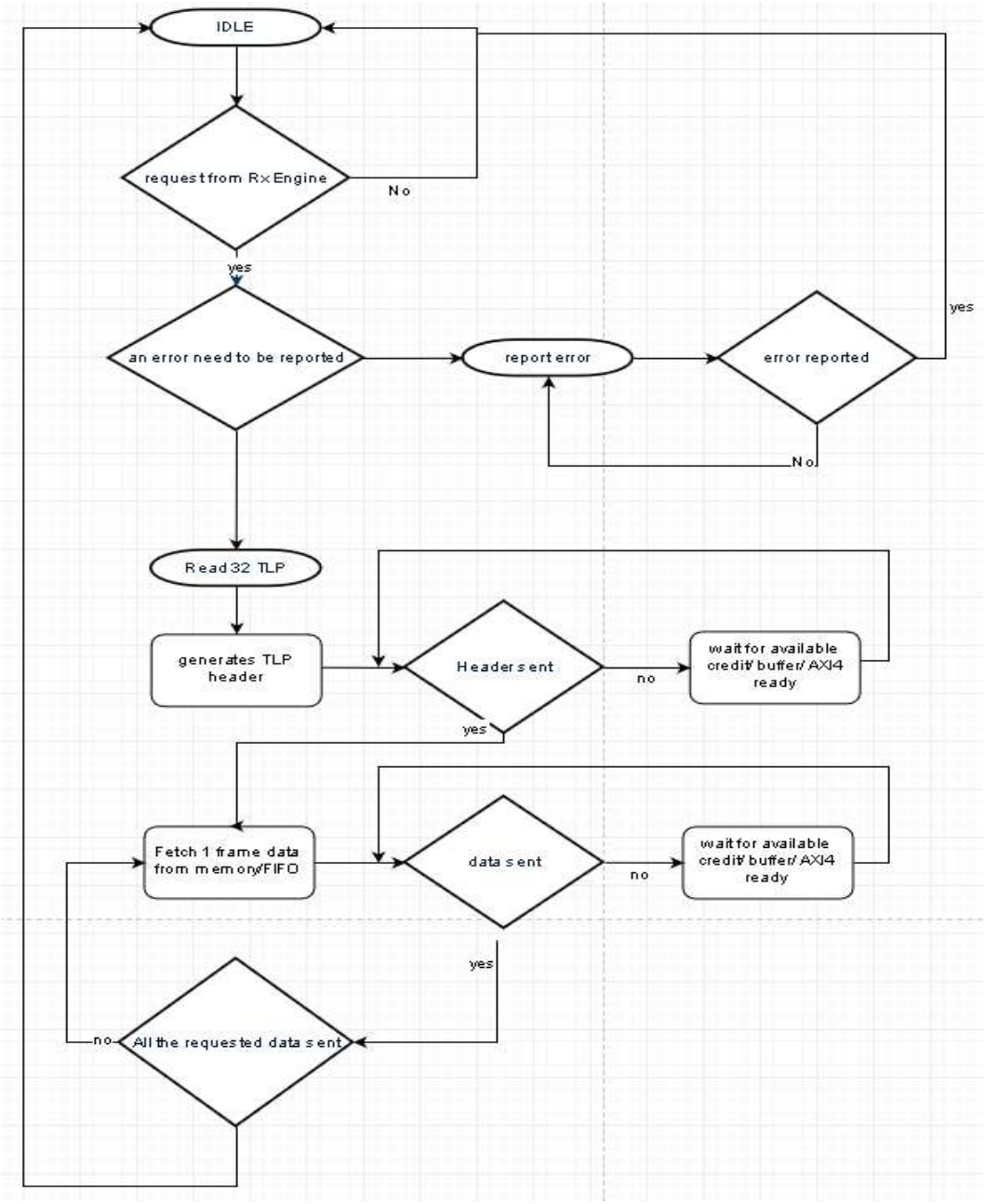


Figure 5.2. Tx Engine State Diagram

One of the most important issues is how to handle the throttling on the transaction layer interface. The system needs to maximize the bandwidth, but which cycle is available for AXI4 transmission depends on the states of both IP core side and FIFO side. For example, if the IP decides to start throttling because there is no available space in the transmit buffer, then even if there is data in the system's stream data FIFO, the Tx engine should stop fetching new data and hold the current data until new space is available. If there is no available data in system's stream data FIFO, even if IP side is available, the Tx engine needs to start source throttle by de-asserting axi_valid signal. Figure 5.3. and Figure 5.4. show the 2 cases of throttles. Note that 2 kinds of throttles are not mutually exclusive. They can overlap with each other, which could become more complicated, as the Figure 5.5. shows.

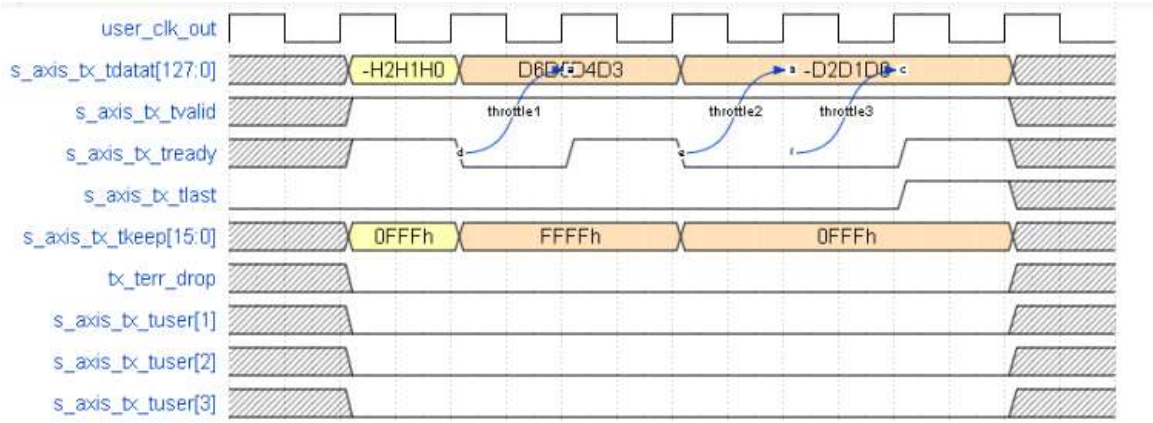


Figure 5.3. Destination Throttle

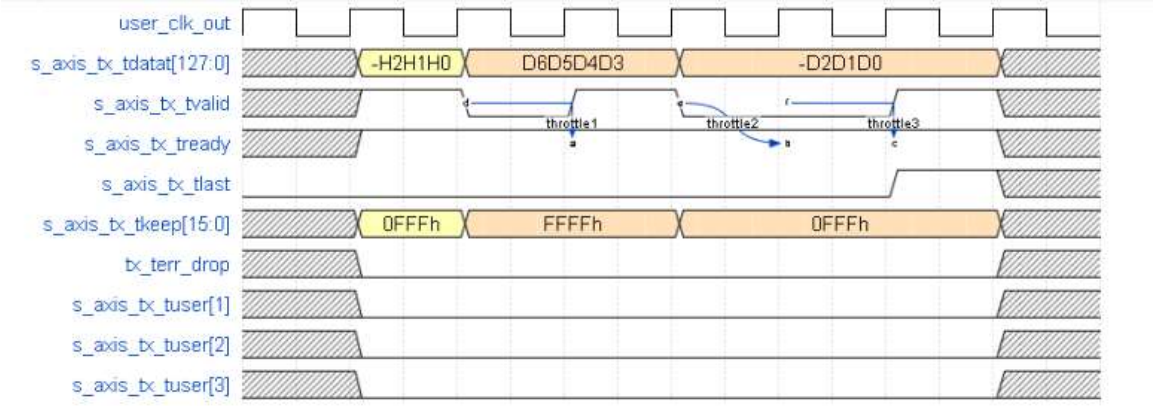


Figure 5.4 Source Throttle

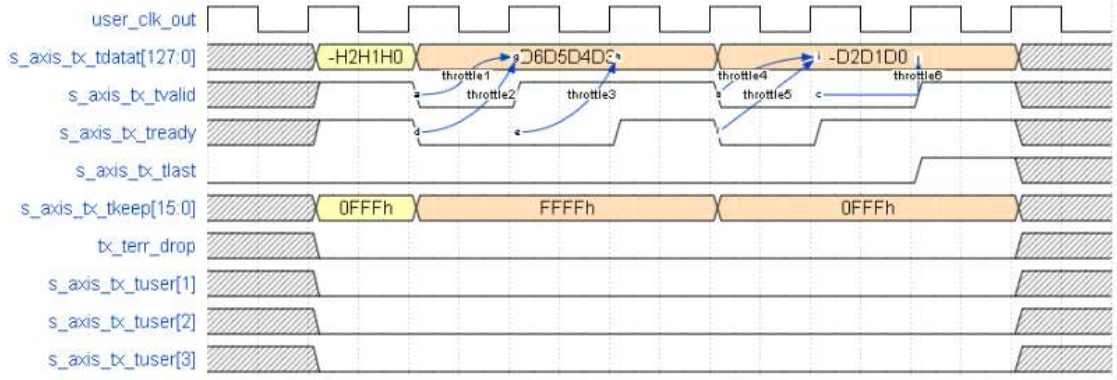


Figure 5.5. Mixed Mode Throttle

To maximize the bandwidth of the system, the logic design should present TLP to the Transaction Layer Interface as long as there is available space. In other words, the logic design should satisfy the following: If the IP starts a Throttle, the logic should hold the current data, axis_valid and axis_tkeep signal, waiting until the next first cycle IP re-assert axis_ready signal. At this same cycle, user logic keeps the holding data and fetches the next data frame.

Challenges come when a FIFO is used for the inbound system stream. Fetching data from FIFO is FIFO read operation. A standard FIFO's timing graph is shown in Figure 5.6., assumes its depth is 4.

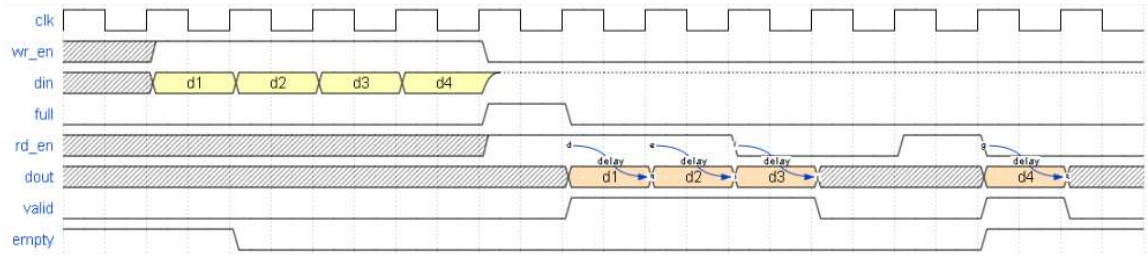


Figure 5.6. Standard FIFO Timing

As shown in Figure 5.6., if a standard FIFO doesn't receive a read enable (rd_en) signal at the clock edge, the data output (dout) is undefined. Only when FIFO receives a rd_en signal, does it present the next data (if FIFO is not empty) at the next cycle. In other words, there is one clock cycle delay before the logic asserts rd_en and data becomes available for use.

Taking the source throttle into account, once a new buffer/credit is available in the IP, the earliest moment of logic asserting rd_en of FIFO can be no earlier than the IP asserting the axis_tready signal, as shown in Figure 5.7., but the actual available data comes at the next clock cycle. From the viewpoint of the IP side, at least one clock cycle of bandwidth is wasted.

It becomes worse when the IP only re-asserts the axis_tready signal for one cycle. Because in this case, when the available data coming from the FIFO is presented to IP at the next cycle, the IP is unable to receive it.

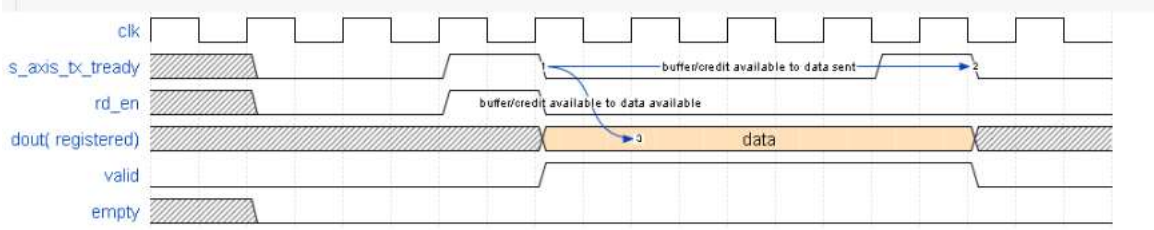


Figure 5.7. Standard FIFO Timing

An extreme case is shown in Figure 5.8. If the IP is never available more than 1 clock cycle and only half of bandwidth is available (in the whole 4 cycles, the IP side is available for 2 cycles) because of some unpredictable Link problems, the actual Link bandwidth usage is only 25 percent. The actual usage is only 50 percent in this case (50 percent available bandwidth, 50% usage of available bandwidth). In this case, the system's bandwidth for a x8 Link of 5.0 GT/s data rate would drop from 32 G bit/s to 8G bit/s (the original bandwidth is 32 Gb/s not 40 Gb/s, because some of the bandwidth is consumed by 8b/10b encoding/decoding and clock recovery/synchronization).

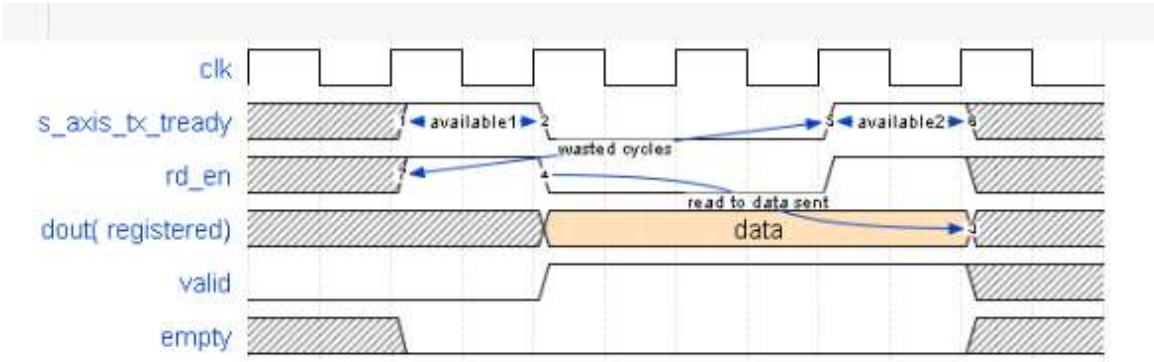


Figure 5.8. An Edge Case of Standard FIFO

FWFT FIFO and Its Use in The System

If using a standard FIFO, the problem can never be solved. This system uses the first-word fall-through (FWFT) to solve the problem.

As its name suggests, the First-Word-Fall-Through (FWFT) FIFO doesn't need an read operation to present the first data to data output bus. Whenever data is available, FWFT FIFO presents it at the data bus (like data just falls through the channel), with the valid signal asserted indicating that data is available. Note that depending on different configurations, the data_valid signal may have different delays related to first write operation.

Figure 5.9. shows the timing of a FWFT FIFO. After first write operation is successful, the valid signal is asserted and the first data is presented at the bus. When first rd_en is detected at the clock edge, FWFT FIFO updates the current first data with the second. Multiple read operation are performed successfully with the exception of the last one, which causes an underflow of the FIFO. Both the underflow and de-asserting of the data_valid signal indicate the error. Note for a FIFO with available depth N, N times of read operation are needed to get all data out (in Figure 5.9., N is 4). This is the same with standard FIFO and FWFT FIFO [18].

FWFT FIFO solves the problem mentioned in the previous section; additional available cycles are wasted. Using destination throttle signal as rd_en signal when FWFT FIFO is not empty updates the current data with next data rather than initiating a read operation with delay in standard FIFO's case. Because the first data is presented on the bus even before rd_en is asserted, first data will be sent whenever new credit is available.

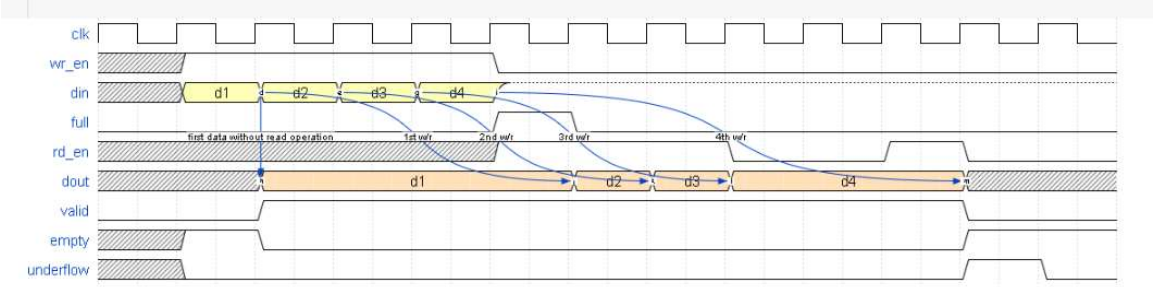


Figure 5.9. FWFT FIFO Timing

Figure 5.10. shows the usage of FWFT FIFO in the system. When IP starts destination throttle by de-asserting axi_ready signal, no read operation is performed but data is presented at the bus. Once the destination throttle stops, first data is sent without any delay and bandwidth waste. The re-asserting of axi_ready is also used as rd_en to update the data bus, and subsequent available cycles remain unaffected.

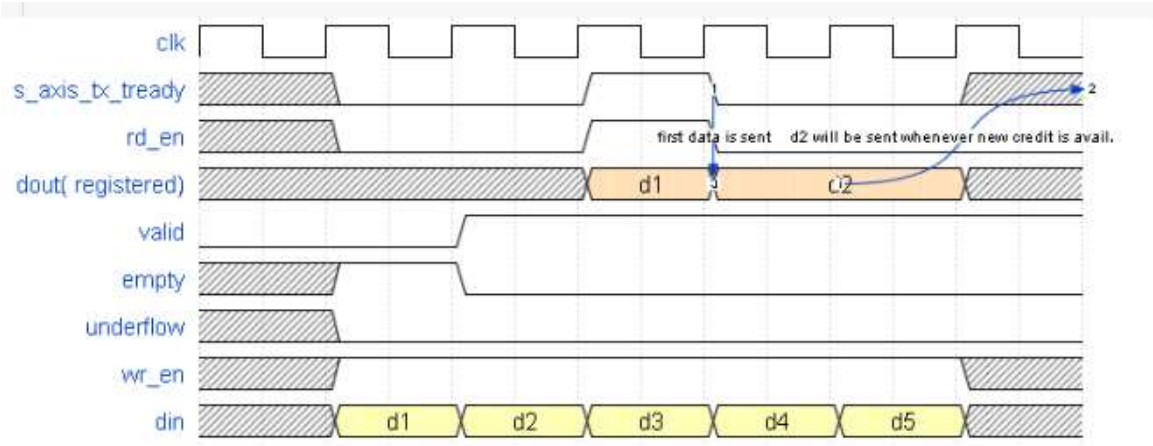


Figure 5.10. FWFT FIFO Timing with Throttle

For the extreme case in previous section, design with FWFT FIFO makes the system's real bandwidth as the same as the available bandwidth. As shown in Figure 5.11., the first data is sent at the first available cycle and FWFT FIFO updates the current

data. When next available cycle comes, updated data is sent. Real bandwidth is 50 percent in this case, same as available bandwidth.

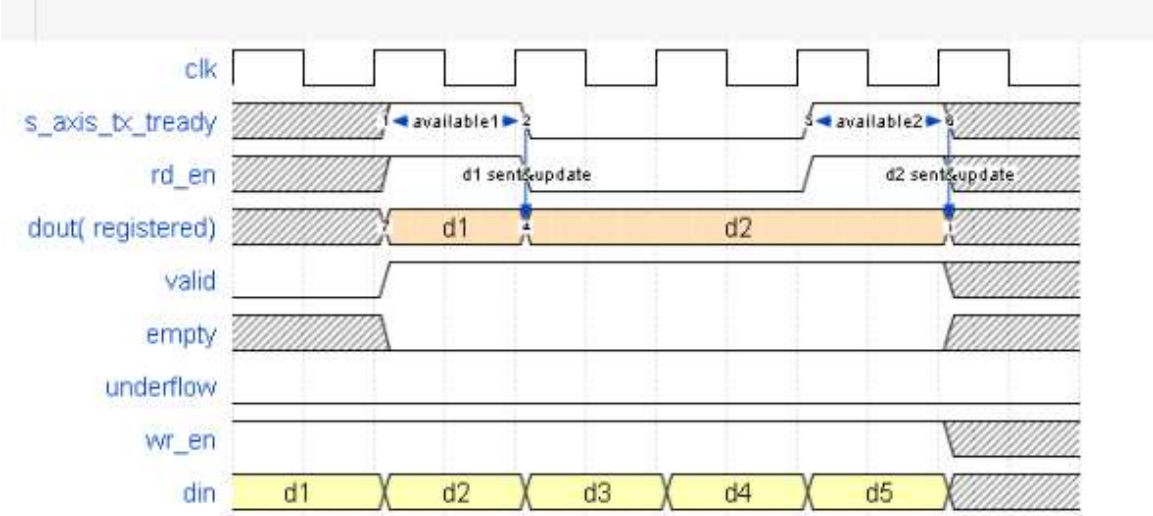


Figure 5.11. An Edge Case of FWFT FIFO usage

Tx Engine State Machine

A normal state machine logic design consists of several major parts, including state encoding, state transition, and output logic. Take the Tx Engine in this system for an example, a normal design is shown as Figure 5.12. The design shown in Figure 5.12. is a standard 2 phase FSM logic design. This is quite clear. But this hardcoded design has 2 inevitable disadvantages:

1. Hardcoded state encoding is not suitable for future extension.
2. Every state needs a unique state name parameter.

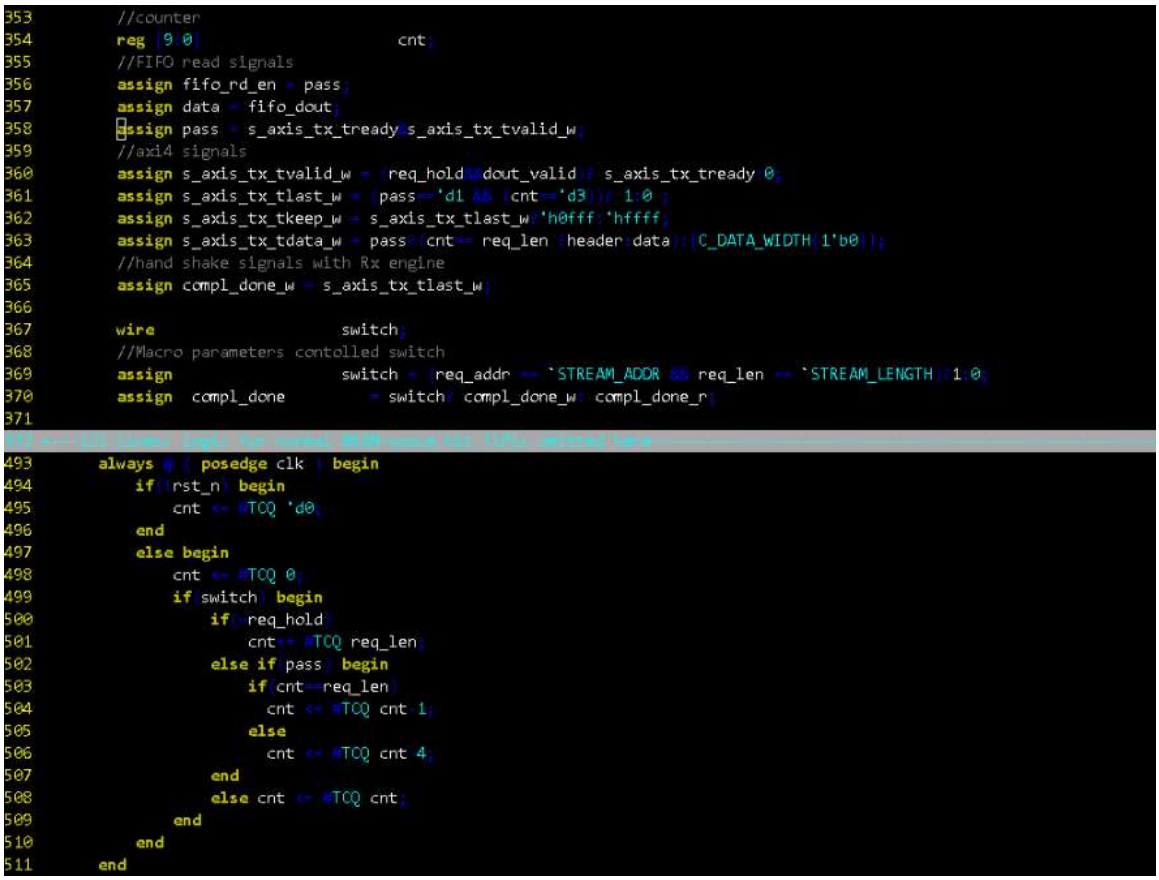
```
104         //state encoding
105         //note if new states are added,
106         //encoding part should also be modified
107         localparam IDLE       = 5'b00001;
108         localparam HEADER     = 5'b00010;
109         localparam DW0_3      = 5'b00100;
110         localparam DW4_7      = 5'b01000;
111         localparam ERROR      = 5'b10000;
112
113         // registers
114         reg [4:0]       st, next_st;
115
116         //state update
117         always @(posedge clk) begin
118           if (!rst_n) begin
119                 st  <= #TCQ IDLE;
120           end
121           else begin
122                 st <= #TCQ next_st;
123           end
124         end
125
126         //state transition and output logic
127         //note if new states are added,
128         //new case branches should also be added
129         always @(*) begin
130                 case (st)
131                         IDLE begin
132                                 //state transistion logic
133                                 if(CONDITION1) next_st = IDLE;
134                                 else if(CONDITION2) next_st = HEADER
135                                 else next_st = ERROR;
136                                 //output drive
137                                 //fifo read side
138                                 if(CONDITION3) begin
139                                         fifo_rd_en = ...
140                                         fifo_dout = ...
141                                 end
142                                 else begin
143                                         fifo_rd_en = ...
144                                         fifo_dout = ...
145                                 end
146                                 //axi4 side
147                                 if(CONDITION4) begin
148                                         axi_data = ...
149                                         axi_keep = ...
150                                         axi_valid =
151                                         axi_tuser = ...
152                                 end
153                                 else begin
154                                         axi_data = ...
155                                         axi_keep = ...
156                                         axi_valid =
157                                         axi_tuser = ...
158                                 end
159                         end
160                         //similar structure for other states
```

Figure 5.12. Tradition FSM Design for Tx Engine

For example, if we need to extend current FSM design from 3DW header + 8DW data payload to 3 DW hear + 128 DW data payload, first we need to add 30 unique parameters for the new states' names (data bus is 128-bit wide, which is 4DW, the added 120 DW data payload needs 30 new states). Then for each new state, we need to add new output logic. In fact, for the PCI Express in this system, the demand could change frequently. From the perspective of software development, tt is unacceptable to re-code huge sections with similar functions every time the demand changes. In other hands, the Tx engine switches to fetch data from FWFT FIFO when TLP request matches specific address and length. When new added states need to drive FIFO reading signal set, which is in another module, it is error-prone.

This Tx Engine adopts a simple but efficient counter-based FSM with Macro defined parameters.

As shown in Figure 5.13., the whole state transition is just counter value updates. The new design has at least 2 advantages: First, it doesn't require any hardcoded state encoding; second, the switch of BRAM and FIFO is controlled by Macro parameter STREAM_ADDR and STEAM_LENGTH. Any demand change only requires modifying the header file at compile time (the header file also contains other configurable system parameters, such as user clock frequency, log on/off switch). FWFT FIFO read signal set is driven only by combination logic, as shown in line 355-363. Because no register is used on the FWFT FIFO read side, the FPGA LUT resource can be saved for other features.



```
353         //counter
354         reg [9:0]                   cnt;
355         //FIFO read signals
356         assign fifo_rd_en = pass;
357         assign data = fifo_dout;
358         assign pass = s_axis_tx_tready&s_axis_tx_tvalid_w;
359         //axi4 signals
360         assign s_axis_tx_tvalid_w = req_hold||dout_valid? s_axis_tx_tready:0;
361         assign s_axis_tx_tlast_w = (pass=='d1 && (cnt=='d3))? 1:0 ;
362         assign s_axis_tx_tkeep_w = s_axis_tx_tlast_w?'h0fff:'hffff;
363         assign s_axis_tx_tdata_w = pass?(cnt== req_len ?header:data):{C_DATA_WIDTH{1'b0}};
364         //hand shake signals with Rx engine
365         assign compl_done_w = s_axis_tx_tlast_w;
366
367         wire                    switch;
368         //Macro parameters contolled switch
369         assign                  switch = (req_addr == `STREAM_ADDR && req_len == `STREAM_LENGTH)?1:0;
370         assign  compl_done          = switch? compl_done_w: compl_done_r;
371
493       always @( posedge clk ) begin
494          if(!rst_n) begin
495             cnt <= #TCQ 'd0;
496          end
497          else begin
498             cnt <= #TCQ 0;
499             if switch) begin
500                if(req_hold)
501                   cnt<= #TCQ req_len;
502                else if pass) begin
503                   if(cnt==req_len)
504                     cnt <= #TCQ cnt+1;
505                   else
506                     cnt <= #TCQ cnt+4;
507                end
508                else cnt <= #TCQ cnt;
509             end
510          end
511       end
```

Figure 5.13. A Counter Based FSM Design with Marco-Defined Parameters

Rx Engine

Compared with the Tx Engine, the Rx Engine's function and design are simpler. The Rx Engine used in the system is based on Xilinx 7 Series FPGAs Integrated Block for PCI Express's example design, with modification to support the system steam data transmission using long length Memory 32 TLPs. This section only briefly discusses the state diagram and important extended feature of the Rx Engine.

If the transaction layer AXI4 interface indicates there is new TLP presented in the data bus, and Rx Engine is not busy, it parses the TLP header and checks the Fmt and Type fields of the header. If it is a Memory 32 write TLP, it parses the destination address field and converts it to the local BRAM address with an appropriate byte mask

set according to the header's Byte Enable fields and AXI4 bus's keep signal. Then it passes the data payload of the TLP, address and mask to BRAM controller interface, finishing the write process. In the whole process of memory write, the Rx Engine keeps in a busy state until BRAM write is successful. Memory write transaction doesn't require completion TLP according to PCI Express SPEC; once write operation is finished, Rx Engine is ready to receive next TLP.

If TLP is a Memory 32 read type, it extracts all the header information from different DWs in AXI4 interface, including Transaction Class , Address Type, Attributes, Tag, Requester ID, Request Length, Byte Enables, and Poisoned TLP Indicator, passing them to the internal bus between Tx Engine and Rx Engine, along with a request signal for handshaking with the Tx engine. It waits for the Tx Engine to complete the read operation by sending completion-with-data TLP to IP. Once it receives the request_complete signal form the Tx Engine, it goes back to IDLE state and waits for the next TLP. The state diagram of Rx Engine is shown in Figure 5.14.

I/O type transaction is not discussed in the section.

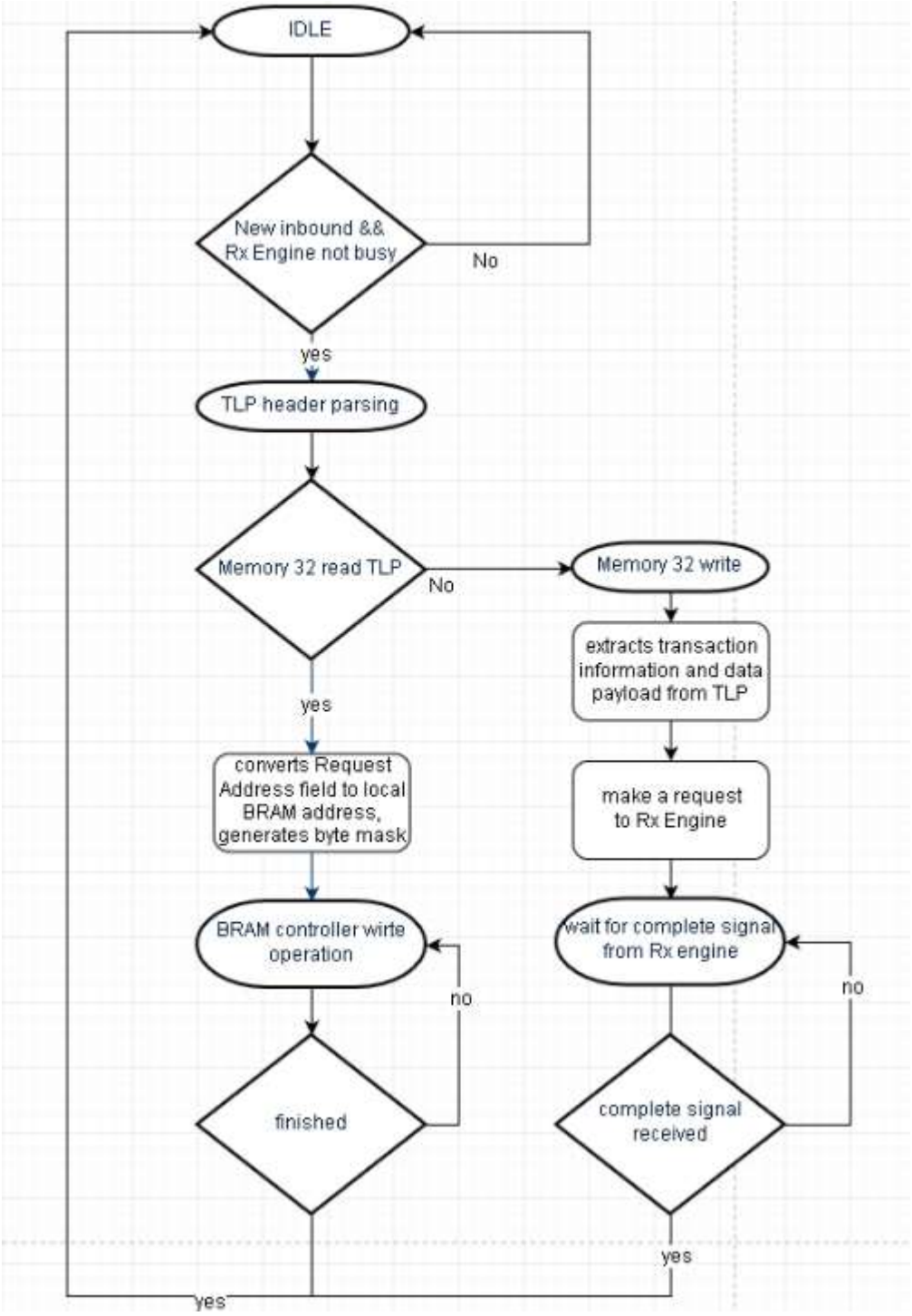


Figure 5.14. Rx Engine State Diagram

CHAPTER SIX

Automated Test and Performance Evaluation

This chapter discusses the testing of the logic design, including an automated test batch program, an abnormal behavior of the Root Complex behavioral model, test flow and result. This chapter gives a performance analysis based on the test result.

Design Testing

Testing Structure

The structure of the typical PC with the PCI Express bus consists of the CPU, main memory, Root Complex, PCI Express Switch (optional) and PCI Endpoint. This system serves as a PCI Endpoint, as shown in Figure 2.2.

Typically, user application uses API functions provided by PCI Express Driver, negotiating lane widths and link speed with PCI Endpoint in initialization process, configuring and allocating main memory space to the endpoint, reading its device information and configuring its register space (including BARs, Configuration/Capability Structures). Once the software initialization and configuration are completed, the user application uses API functions to communicate with the Endpoint by sending and receiving TLPs. Any on-board test requires the cooperation of user application and Endpoint logic design, with operating system and system driver as intermediate levels, as shown in Figure 6.1.

Before on-board testing, logic design needs to be tested separately to expose and separate bugs. In this case, a behavioral model is needed in simulation to serve as the functionality of the Root Complex.

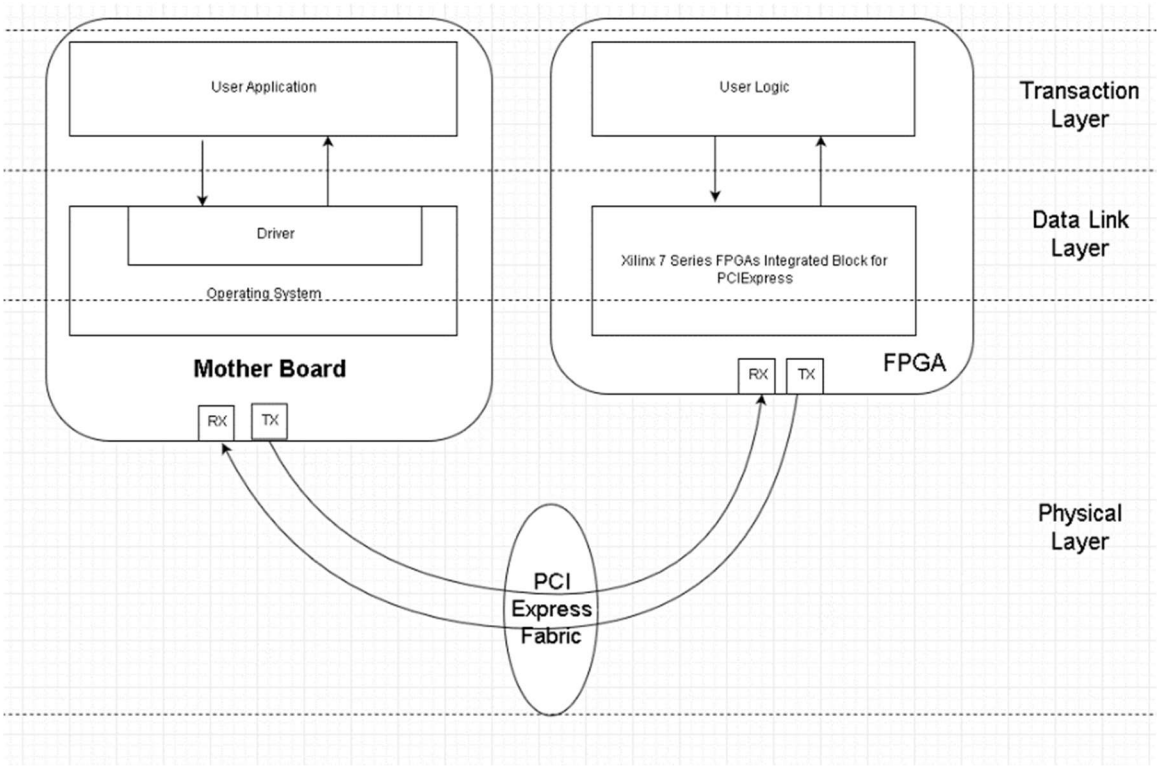


Figure 6.1. PCI Express Structure in PC

The PCI Express Root Port Model provided by XILINX is used in testing. The Root Complex model provide provides API for system initialization, Type0/1 Configuration, Memory 32/64 TLP Transaction, I/O Transaction and Message Transaction, in the form of Verilog/System Verilog code. Thus, significant time could be saved by verifying the functionality of TLP Layer user logic, rather than developing the whole test infrastructure to simulate the behavior of Root Complex and PCI Fabric. In fact, in modern logic design/verification industry, there is a market segment focusing exclusively on providing PCI Express Verification IP (PCI Express VIP). The Root

Complex Model provided by XILINX is not as powerful as industrial-level VIP, but it is free and covers the basic usage and testing of this system.

The test environment structure is shown in Figure 6.2., the testing program and logging system is separated from Root Complex model. The testing program invokes APIs provided by the Root Complex model, communicates with the Xilinx 7 Series FPGAs Integrated Block for PCI Express by PCI Express Fabric (LVDS pairs), user logic design lies above IP. With correct configuration, user logic is tested under the same conditions as on-board testing.

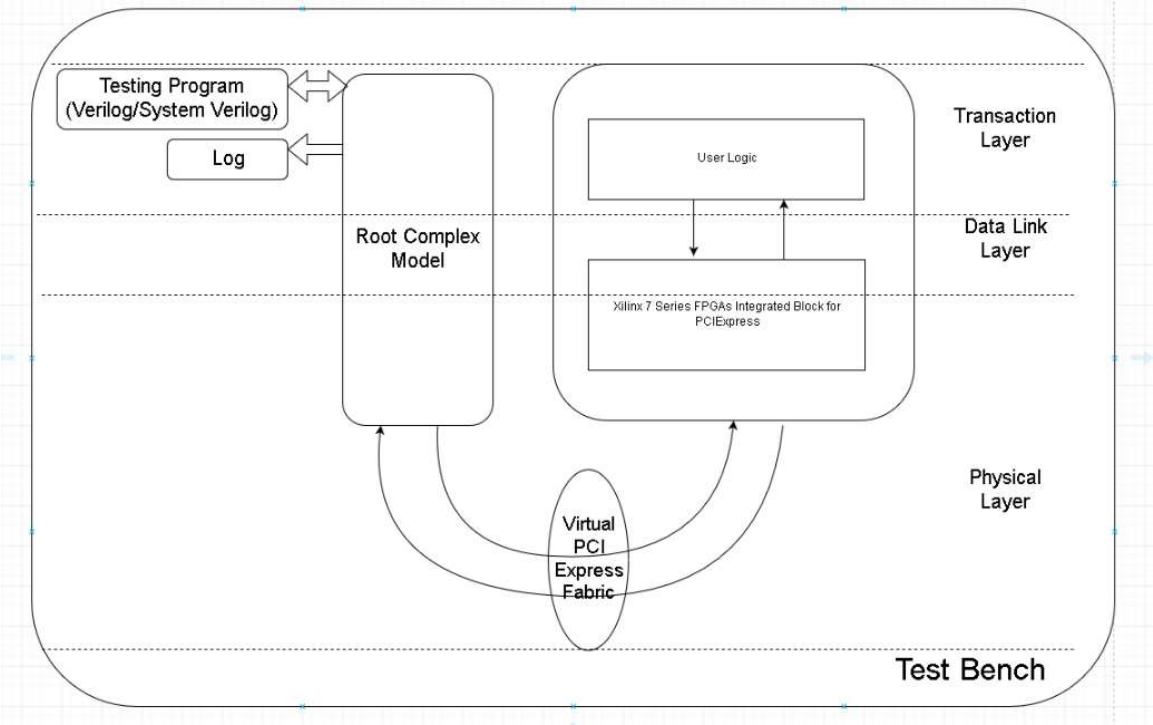


Figure 6.2. Test Environment Structure

Root Complex Model and Test Flow

The Root Complex model consists of 4 parts: the tx block, the rx block, the common block and the dsport block. The common block provides shared logic for the tx block and the rx block. The tx block send out the request based on the user test program, in the form of TLPs. The rx block receives and checks the inbound TLP from the dsport block [17]. The dsport is responsible for all the functionality below Transaction Layer. The hierarchy of Root Complex model is shown in Figure 6.3.

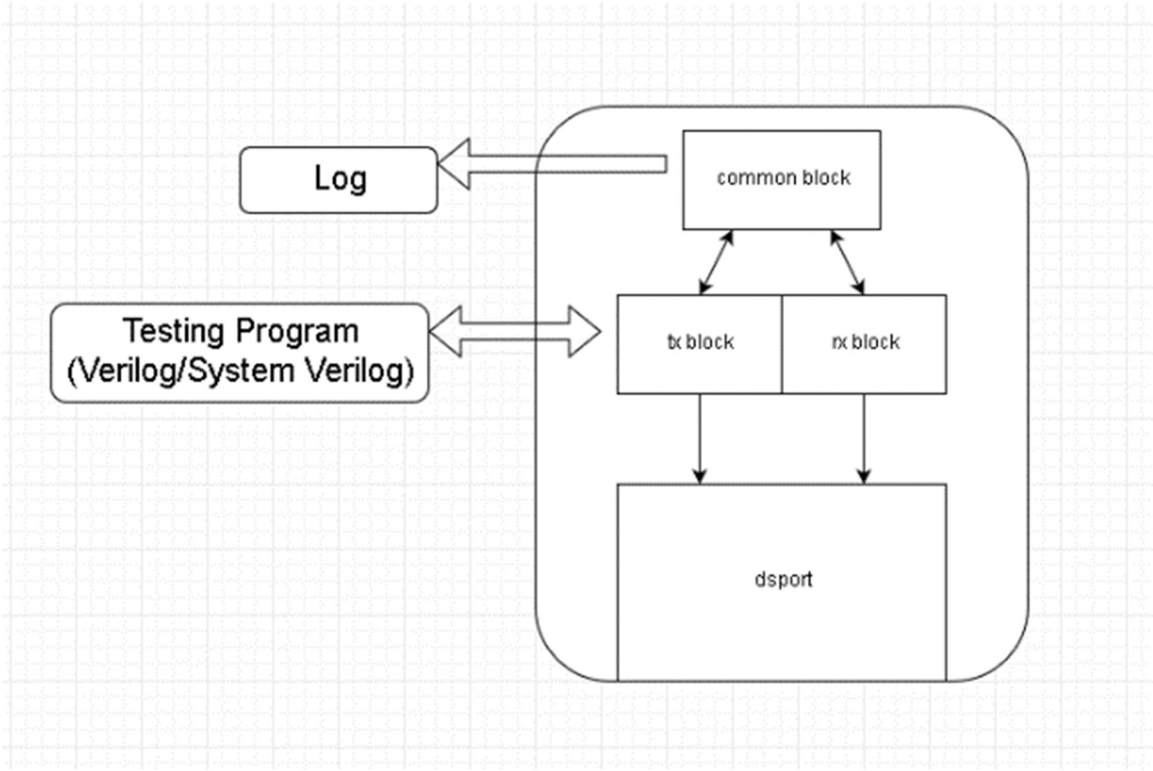


Figure 6.3. Root Complex Model Structure

Details about the Root Complex model can be found in Xilinx Datasheet. This section only covers the part which is important for test implementation.

The testing program is sent to the tx block by invoking APIs. More details about APIs can be found in Xilinx datasheet. The usage of APIs conforms to [16].

The whole test flow is shown in Figure 6.4.

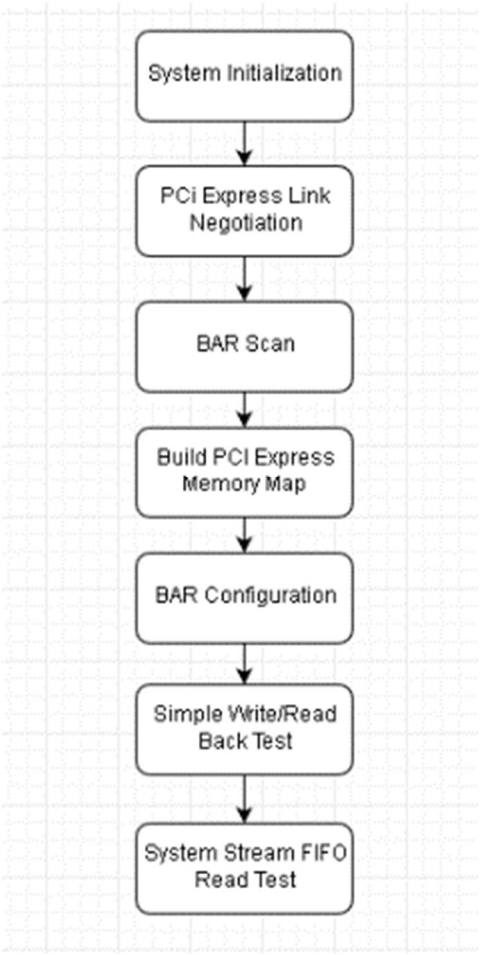


Figure 6.4. Test Flow

Logging and Automated Test

Logging. One of the basic debugging and testing methods is reading the timing graph. The timing graph contains all the information of logic design on the RTL level. But the timing graph is not a good fit for integrated design and industrial level automated testing. For example, in this testing, system initialization and BAR initialization are finished by APIs. Because the time consumed by those two steps is at millisecond level,

it is impractical to include all the information in two steps in the timing graph, for at least two reasons:

1. Once testing environment is set up, for the whole testing cycle, those two parts are very likely to remain the same.
2. To present millisecond level information in nanosecond level(main clock of the system is 250 MHz) , the simulation tool requires a large amount of memory consummation.

As mentioned in the previous section, the focus of testing is on the Transaction Layer. It involves a lot of rule checking and packet parsing. Using the timing graph to check all those functionalities is very time consuming and inefficient. Modern logic verification introduces the Transaction Layer Modeling to give a high-level abstraction of the function of any logic design. For this design, an appropriate abstraction focuses on the request and response transmitted between the RC and the Endpoint. For example, the RC sends a Memory 32 TLP and the Endpoint responds with a Completion with Data TLP. In this case, the verification should focus on the information contained in TLPs, such as request length, data payload, request sent time and response received time, rather than on the bit level wiggling on the timing graph.

The logging interface of the RC model satisfies this need.

A part of the simulation log is shown in Figure 6.5. At 116469411 ns , the RC starts test targeting at BAR 0 of Endpoint. 2 outbound TLPs are sent and 1 inbound TLP is successfully received and checked. Part of Payload is also shown.

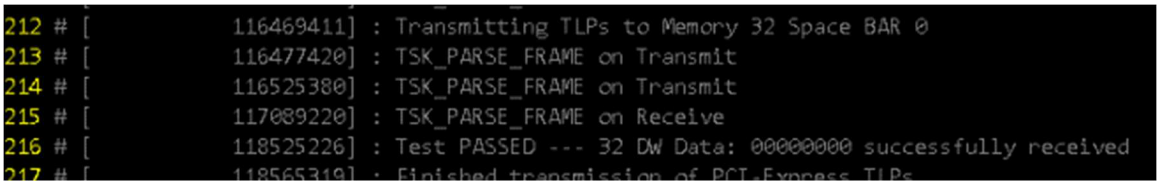


Figure 6.5. Simulate Log Example

By the use of the logging interface, large amounts of time and resources could be saved from reading a low bit wiggling timing graph. If any error happens, the log helps to locate the error faster, as shown in Figure 6.6.

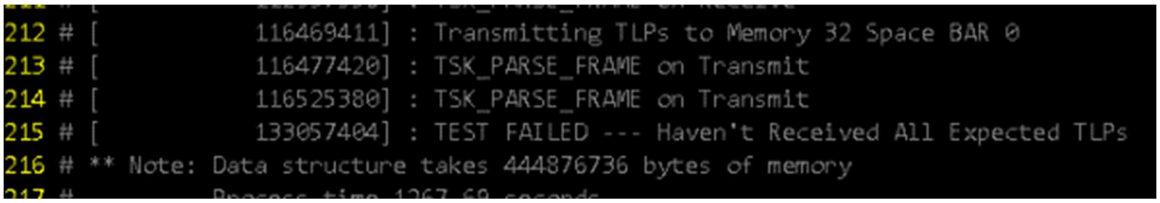


Figure 6.6. Error Log Example

If detailed information of TLP is wanted, tx log and rx log provide all the TLPs in a simulation. Figure 6.7. shows a Type 0 Configuration Write TLP sent at 112469000 ns.

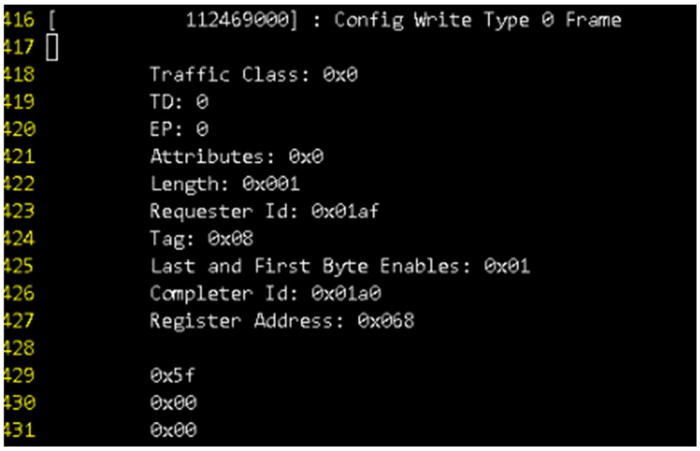


Figure 6.7. Type 0 Configuration TLP Write Tx Log

Automated Testing. With logging, only logging files are checked after simulation for most of the time. Therefore, invocation of the simulation tool with GUI is also not needed for most of the time, for two reasons:

1. The GUI's most important function is acting as a utility where users can check bit level events.
2. The GUI mode requires too much manual operation, such as running/stop control, wave format configuration/load, zoom in/out.

In this test, automated batch programs are developed for every step, including:

1. compile batch, which compiles source code in an incremental way, compile warning and error are given in the compile log.
2. simulate batch, which tests the design in non-GUI mode, tx log, rx log and simulate log are provided after finish
3. view batch, which provides the viewing the simulation in GUI mode, for debugging
4. simulate with GUI batch, which tests the design in GUI mode, providing a real time view of wave
5. wave file convert batch, which converts the simulation result file to VCD format (a more widely supported wave form format).

Parts of the batch program is shown in Figure 6.8.

Figure 6.8. Automated Testing Batch Program Code Example

Abnormal Behavior of Root Complex Model and Solution

In order to maximize the bandwidth usage, data payload of the TLPs should be set to MAX_PAYLOAD_SIZE of Device Capabilities/Control Registers. For x8 Link of 5.0 GT/s PCI Express Fabric used in this design, 512 is the maximum supported value.

The Root Complex uses this maximum value as request length as long as it is possible. Based on this configuration, the simulation gives the error that requested TLP hasn't been received, shown in Figure 6.6. More detailed information is given by timing graph in Figure 6.9.

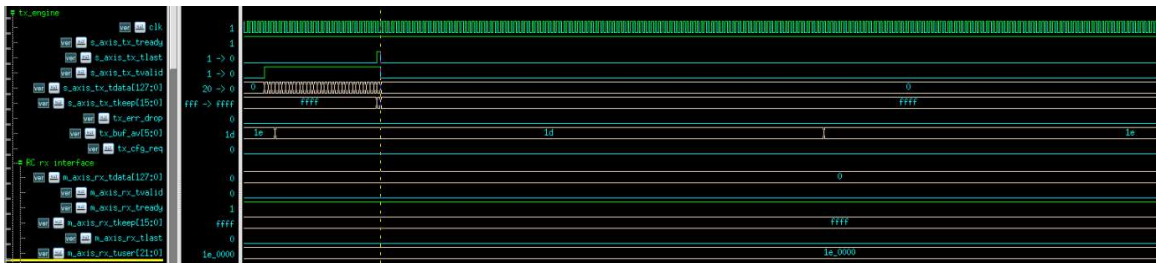


Figure 6.9. 128 DW Memory 32 Read Request Error Timing

As shown in Figure 6.9., the user logic provides a TLP with a 128 DW payload to the Transaction Layer Interface. Even though the transmission is good on the Interface, the IP core allocates a new buffering space for this TLP, which causes the tx_buf_av value drop for ‘h1e to ‘h1d. But after the TLP is received by Endpoint IP, tx_buf_av get reset to ‘h1e, without either the successful transmission to RC side or the assertion of tx_err_drop signal (which indicates there is PCI Express rule violation in this TLP causing the drop). The value of the Device Capabilities Register of both the Endpoint and the RC is shown in Figure 6.10. Bit mapping of the register is shown in Figure 6.11.

At the Endpoint IP side, both bit location 14:12 and 7:5 has value 000, which indicate the supported Max_Read_Request_Size and Max_Payload_Size are both 512 Byte. But at the Root Complex model side, bit location 14:12 has a value of 010, while 7:5 has a value of 000, which means the RC side only supports up to 128 Bytes payload in a TLP.

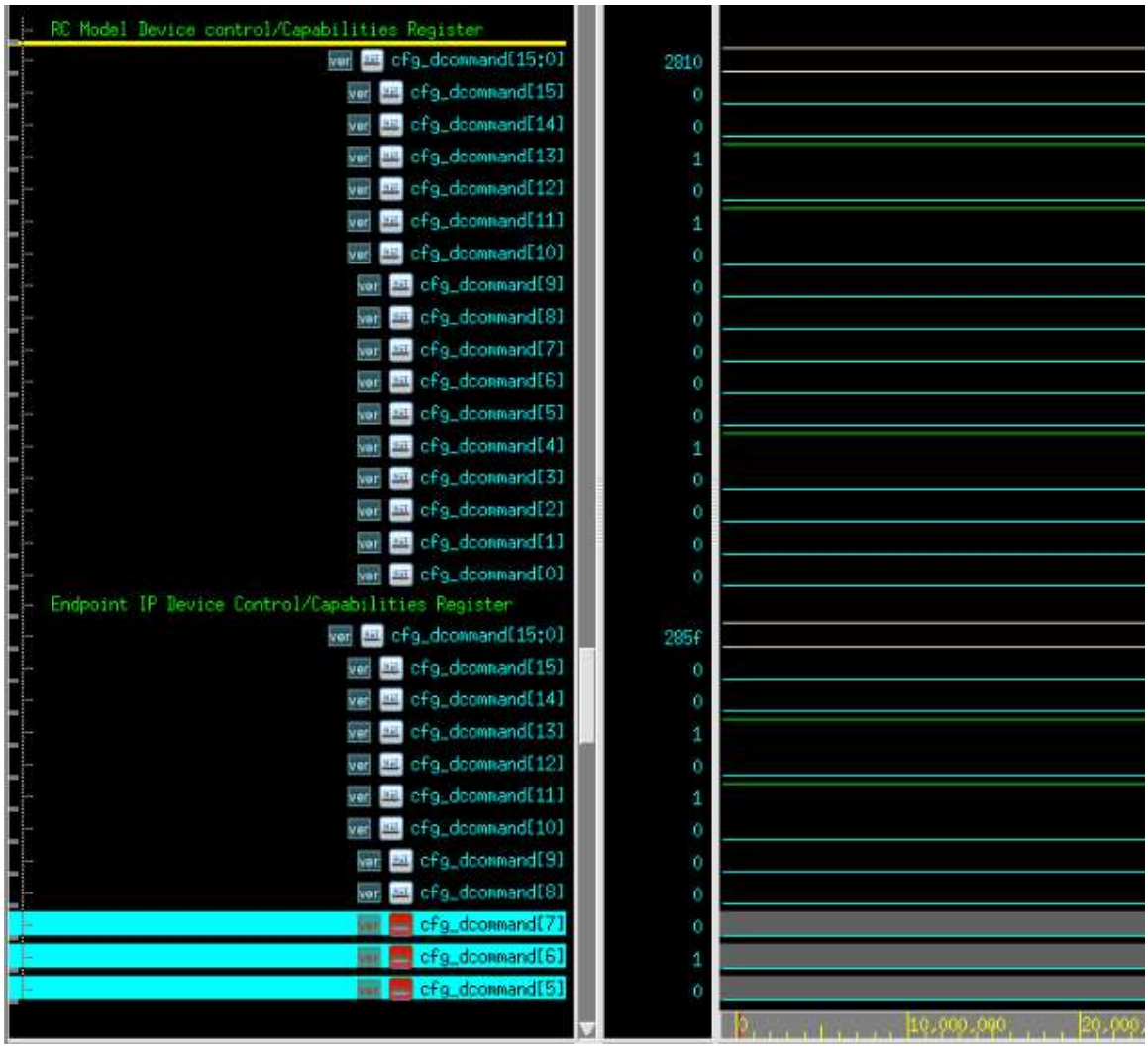


Figure 6.10. Device Control/Capabilities Registers Value

Bit	Name
cfg_dcommand[15]	Reserved
cfg_dcommand[14:12]	Max_Read_Request_Size
cfg_dcommand[11]	Enable No Snoop
cfg_dcommand[10]	Auxiliary Power PM Enable
cfg_dcommand[9]	Phantom Functions Enable
cfg_dcommand[8]	Extended Tag Field Enable
cfg_dcommand[7:5]	Max_Payload_Size
cfg_dcommand[4]	Enable Relaxed Ordering
cfg_dcommand[3]	Unsupported Request Reporting Enable
cfg_dcommand[2]	Fatal Error Reporting Enable
cfg_dcommand[1]	Non-Fatal Error Reporting Enable
cfg_dcommand[0]	Correctable Error Reporting Enable

Figure 6.11. Device Control/Capabilities Register Bit Map

The reason that the Endpoint IP discards the successfully-received TLP in core buffering space is that the Root Complex model, on the other side of the link, doesn't support a payload of 512 Bytes. The Endpoint IP clears the buffering space and restores the credit after discarding the TLP.

After modifying the read request length to 128 Bytes, transmission is completed successfully, as shown in Figure 6.12. Note that the Endpoint IP restores the credit after the RC receives the TLP.

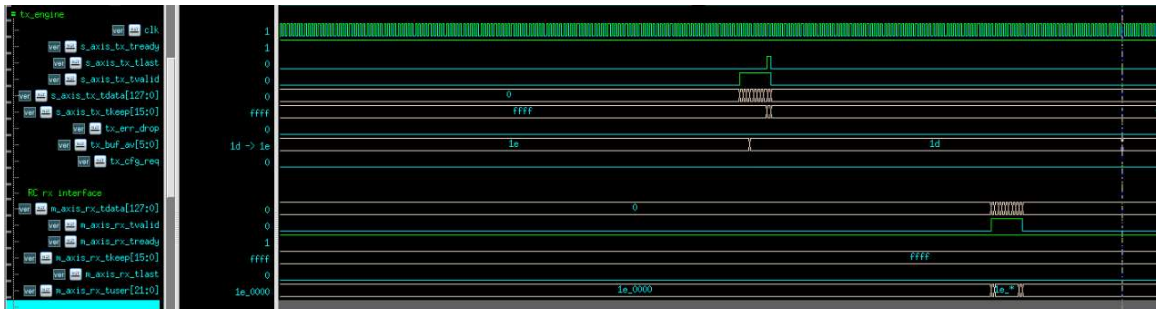


Figure 6.12. 32 DW Memory 32 Read Test Result

Performance Analysis

Because the Root Complex model provided by Xilinx doesn't provide any mechanism to configure its Device Control/Capabilities Register, performance analysis is based on the test results of 32 DW Memory TLPs, and is extended to 128 DW case.

A complete data communication consists of 4 parts, as shown in Figure 6.13:

1. Transmission time from RC sending out Read Request to Endpoint receiving it, notated by t_1.
2. Time for Rx Engine processing inbound TLP, notated by t_2.
3. Time for Tx Engine fabricating outbound Completion with Data TLP, notated by t_3.
4. Transmission time from Endpoint sending out TLP to RC receiving it, notated by t_4.

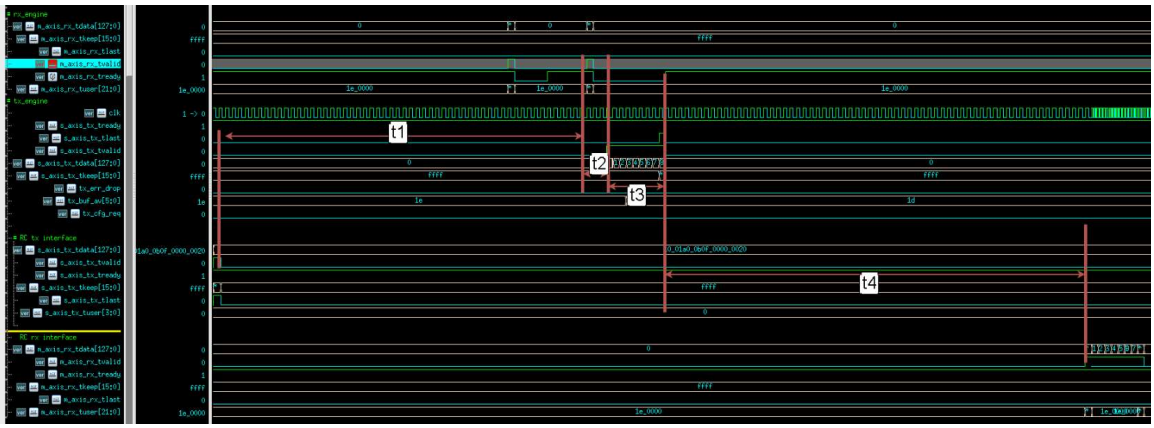


Figure 6.13. Complete Data Communication Cycle

t_1 and t_4 are fixed values for a PCI Express Link. For the Rx engine, request processing time t_2 for any 3DW header Memory 32 Read TLP is also a fixed value, if there is available core buffering space. Assuming there is always available data in system stream FWFT FIFO, t_3 is the sum of 3DW header processing time, notated by t_{3h},and processing time for every 4 DW data, notated by t_{3d}. The real Bandwidth of the system, notated by b_{real}, can be calculated by:

$b_{real} = \frac{p\_size * n}{t_1+(t_2+t_{3h}+t_{3d}*N)*n+t_4}$, N is the number of 4DW in payload, n is the maximum read TLP number of the credit/core buffering space, which is 32 in this design

According to the timing graph, $t_1 = 227\ ns$, $t_2 = 12ns$, $t_{3h} = t_{3d} = 4\ ns$, $t_4 = 256\ ns$. Note that in the real application, software is able to configure MAX_PAYLOAD_SIZE and MAX_READ_REQUEST_SIZE of Root Complex Control/Capabilities Register to 128, in this case N is 32.

$$b_{real} = \frac{128\ DW * 32}{227\ ns+(12\ +4+\ 4*32)\ ns\ *32+256\ ns} = 24.093\ Gbit/s$$

24.093 Gb/s is the real bandwidth in case of a 32 consecutive 128 DW TLPs burst transmission. Note that when the Tx engine sends out the first Completion with Data TLP,

it continues to process the following 31 TLPs, notated by $t_{3following}$. $t_{3following}$ overlaps with t_4 and $t_{3following} > t_4$, which means RC receives the first Completion with Data TLP when Tx Engine is still processing the following read request. In this case, the RC doesn't need to receive the last expected TLP before making a new read request, because new credit is available after receiving first TLP. Therefore,

$$b_{real} = \lim_{n\to\infty}\left(\frac{128\ DW * n}{227\ ns\ \ (12 +4+ 4*32)\ ns * n+256\ ns}\right) = 26.491\ Gbit/s$$

CHAPTER SEVEN

Conclusion

Summary

The logic design proposed by this thesis satisfies two key concerns of the Phase-II pCT scanner hardware upgrade.

First, the speed limit of the Ethernet link between 'event-builder' FPGA and DAQ Computer. It increases the link speed from 800 Mbit/s to 26.491 Gbit/s, by about 33 times.

Second, the redesign goal of increasing the speed of the preamplifier and shaping amplifiers by about a factor of four, such that the signal peaks in about 50 ns instead of 200 ns. The logic is able to handle signal peaks in 50-ns level because it is operated at a clock cycle of 4 ns (250 MHz). The link delay between the logic and root complex is 256 ns (discussed in chapter six) and it has 32 TLPs' credit. The amortized delay is 8 ns. This exceeds the required factor of four. It greatly reduces the pileup probability, especially when running the system with a pencil beam, and can be accomplished by increasing the sizes of some of the transistors, especially the large input transistor, as well as the currents.

In conclusion, the transmit-received-engine based logic design proposed by this thesis works at the PCI Express Transaction Layer in collaboration with Xilinx 7 Series FPGAs Integrated Block for PCI Express. By automated testing and results evaluation, the new design can speed up the original Ethernet link speed by a factor of 33, At the same time, supports the needs of the new signal peaks in 50 ns.

Future Directions

This thesis has explored the Design and Automated Testing of the PCI Express Interface for Proton Computed Tomography Detectors. Future work includes the followings.

First, adding buffer management module in RX Engine. Rx Engine in Current design starts destination throttle when Tx engine is processing the Memory Read Request. A buffer management module will give the Rx Engine the ability to keep receiving inbound TLPs simultaneously with Tx engine' processing. A better real bandwidth could be achieved in this way. Strongly Ordering is used in this design, which matches the sequential work flow of Rx Engine. If Relaxed Ordering or ID-Based Ordering is required in the future, buffer management module must be added to Rx Engine, because TLP s are not parsed by the order in which they arrive in those cases. The buffer management module in the current RX Engine and switch from Strongly Ordering to ID-Based Ordering could satisfy the future needs of the bandwidth improvement.

Second, root Complex Model provided by Xilinx has 2 problems: it only supports up to 128 Bytes MAX_PAYLOAD_SIZE; and multithreads Request Sending is not supported. To perform functional coverage analysis with UVM, those 2 problems must be solved. The scanner is used in medical treatments, which puts a high demand in reliability. The use of UVM and full functional/code coverage could prove its reliability in a more quantitative way.

Third, current design is the best fit for stream transmission of large chunks of data, which is generated by the Proton Computed Tomography Detectors. For non-consecutive transmission of small sized data, PCI Express Message Transaction is a better choice,

because messages use in-band communications and an independent Transaction Interface of Xilinx 7 Series FPGAs Integrated Block for PCI Express core [17]. The future needs of non-delay-sensitive small sized data communication between Data Acquisition (DAQ) Computer and event-builder FPGA, which could be reporting DAQ's status to the FPGA, can be handled by adding message type TLPs support to the design, without sacrificing the link performance of Proton Computed Tomography Detectors.

BIBLIOGRAPHY

[1] American Cancer Society, "Cancer Facts & Figures 2018," Atlanta: American Cancer Society, 2018.

[2] Vladimir A. Bashkirov, Robert P. Johnson b, Hartmut F.-W. Sadrozinski, Reinhard W. Schulte, "Development of proton computed tomography detectors for applications in hadron therapy," *Nuclear Instruments and Methods in Physics Research*, vol. 809, pp. 120–129, 2016.

[3] R.P. Johnson, *A Fast Experimental Scanner for Proton CT: Technical Performance and First Experience with Phantom Scans*, IEEE Trans. Nucl. Nl. Sci. 63-1, 2015.

[4] Plautz TE, Bashkirov V, Giacometti V, Hurley RF, Johnson RP, Piersimoni P, Sadrozinski HF, Schulte RW, Zatserklyaniy A, *An evaluation of spatial resolution of a prototype proton CT scanner*, Med Phys, Dec, pp. 43-55, 2016.

[5] Sadrozinski HF, Geoghegan T, Harvey E, Johnson RP, Plautz TE, Zatserklyaniy A, Bashkirov V, Hurley RF, Piersimoni P, Schulte RW, Karbasi P, Schubert KE, Schultze B, Giacometti V, *Operation of the Preclinical Head Scanner for Proton CT*, Nucl Instrum Methods Phys Res A. Sep., pp. 394-399, 2016.

[6] Bashkirov VA, Johnson RP, Sadrozinski HF, Schulte RW, *Development of proton computed tomography detectors for applications in hadron therapy*, Nucl Instrum Methods Phys Res A, Feb., pp. 120-129, 2016.

[7] Bashkirov VA, Schulte RW, Hurley RF, Johnson RP, Sadrozinski HF, Zatserklyaniy A, Plautz T, Giacometti V,*Novel scintillation detector design and performance for proton radiography and computed tomography*, Med Phys, Feb, pp.664-674, 2016.

[8] Hurley RF, Schulte RW, Bashkirov VA, Coutrakon G, Sadrozinski HF, Patyal B, *The Phase I Proton CT Scanner and Test Beam Results at LLUMC*, Trans Am Nucl Soc, pp. 63-66, 2012.

[9] Plautz T, Bashkirov V, Feng V, Hurley F, Johnson RP, Leary C, Macafee S, Plumb A, Rykalin V, Sadrozinski HF, Schubert K, Schulte R, Schultze B, Steinberg D, Witt M, Zatserklyaniy A, *200 MeV proton radiography studies with a hand phantom using a prototype proton CT scanner*, IEEE Trans Med Imaging, Apr, pp.33-37, 2014.

[10] Sadrozinski HF, *Particle Detector Applications in Medicine*, Med Phys. Nucl Instrum Methods Phys Res A, Dec, pp. 21-23, 2013.

[11] Volz L, Piersimoni P, Bashkirov VA, Brons S, Collins-Fekete CA, Johnson RP, Schulte RW, Seco J, *The impact of secondary fragments on the image quality of helium ion imaging*, Phys Med Biol, Oct, pp 63-82, 2018.

[12] B. Schultze, P. Karbasi, V. Giacometti, T. Plautz, K.E. Schubert, R.W. Schulte, "Reconstructing highly accurate relative stopping powers in proton computed tomography," *Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*, IEEE, 2015.

[13] Hurley RF, Schulte RW, Bashkirov VA, Wroe AJ, Ghebremedhin A, Sadrozinski HF, Rykalin V, Coutrakon G, Koss P, Patyal B, *Water-equivalent path length calibration of a prototype proton CT scanner*, Med Phys, May, pp. 39-44, 2012.

[14] Giacometti V, Guatelli S, Bazalova-Carter M, Rosenfeld AB, Schulte RW, *Development of a high resolution voxelised head phantom for medical physics applications*, Phys Med, Jan, pp. 182-188, 2017.

[15] Johnson RP, Dewitt J, Holcomb C, Macafee S, Sadrozinski HF, Steinberg D, *Tracker Readout ASIC for Proton Computed Tomography Data Acquisition*, IEEE Trans Nucl Sci. 60-1, 2013.

[16] PCI SIG, *PCI Express® Base Specification Revision 3.0*, PCI SIG, 2010. [Online]. Available: https://pcisig.com.

[17] XILINX, "7 Series FPGAs Integrated Block for PCI Express," LogiCORE IP Product Guide, Dec, 2018.

[18] XILINX, "FIFO Generator," LogiCORE IP Product Guide, Oct, 2017.