DESIGN AND VERIFICATION OF APPLICATION AND TRANSACTION LAYERS OF PCI-EXPRESS

A Major Project submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF ENGINEERING
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

Delhi University, Delhi

Submitted By
ALLAMUDI SREEKANTH (16/E&C/04)
Delhi University Roll no . 8721

Under the guidance of
Mr.O.P.VERMA
Assistant Professor
Department of Electronics & Communication Engineering
DCE,Delhi

Department of Electronics & Communication Engineering
DELHI COLLEGE OF ENGINEERING
Bawana Road, Delhi - 110042

JUNE -2006

CERTIFICATE

This is to certify that this thesis work entitled "**DESIGN AND VERIFICATION OF APPLICATION AND TRANSACTION LAYER OF PCI-EXPRESS**" is the bonafide work carried out by **ALLAMUDI SREEKANTH** (16/E&C/04, D.U.Roll no 8721) who carried out the major project work under my supervision, and submitted in partial fulfillment of the requirements for the award of the degree of Master of Engineering, during the year 2004-2006.

Mr.O.P.VERMA
Assistant Professor,
Dept of ECE
DCE,DELHI

Acknowledgements

Behind every achievement of a student lies the unflinching effort of his teachers without who, as students we could never know the liveliness of hardwork.And to this day and to the day we certainly feel it as our worldly pleasure to get living to this day of thinking them.

I would like to express my gratitude and sincere thanks to our project guide **Mr.O.P.VERMA** Assistant Professor ,department of Electronics & communications engineering, Delhi College of engineering for his esteemed guidance and incessant support given in presenting this project report successfully.

I would thank our head of the department **Prof.Asok Bhattacharyya** for his kind co-operation in bringing out this project work successfully.

I also take this opportunity to express my deep sense of gratitude and sincere thanks to the staff of our Department, **Dr.Kulkarni ,Mr Raghav Mrs Indu ,Mrs Rajeshwari pandey,** for their unstinted co-operation.

I would also like to thank **Mr.Jitendra Puri** and his team of nSys Design Systems Pvt Ltd., for providing opportunity to work on the project .

I am equally grateful and indebted to our beloved Principal **Dr.P.B.Sharma** for providing us all the pre-requisite facilities.

ALLAMUDI SREEKANTH
16/E&C/04
D.U.Roll no 8721

Abstract

With ever-increasing network traffic, bottlenecks are Inevitable in the existing parallel, multi-drop architecture of the Peripheral Component Interconnect (PCI) bus and its second-generation version, the PCI Extended (PCI-X) bus. Today, those bottlenecks can be alleviated with the much higher performance of the third generation PCI Express architecture, which uses a 2.5-GHz clocked serial Input/Output (I/O) structure to provide higher bandwidth and far better scalability than its predecessor I/O architectures. PCI Express is a layer based protocol classified as Transaction layer, data link layer, physical layer.

This thesis provides an overview of the new PCI Express bus architecture and explains an approach to design the Transaction Layer and Application layer of PCI-Express also gives a brief verification strategy to verify serial buses, Implementation is done using Verilog – HDL .

CONTENTS

List of Figures

List of Tables

Chapter 1
Introduction

Peripheral Component Interconnect (PCI) Express is a scalable, standards-based, high-bandwidth I/O interconnect technology incorporating recent advances in high-speed, point-point interconnects. PCI Express architecture is Compatible with the PCI addressing model, load-store architecture with a flat address space, software interfaces are maintained to ensure that all existing applications and drivers operate unchanged. Where as parallel bus implementation of PCI is replaced by a highly scalable, fully serial interface in PCIe. PCI Express provides significantly higher performance, reliability, and enhanced capabilities—at a lower cost—than the previous PCI and PCI-X standards.

PCI Express Architecture is specified in layers. It is classified into three layers namely Transaction layer, data link layer, and physical layer. PCI Express configuration uses standard mechanisms as defined in the PCI Plug-and-Play specification. The software layers will generate read and write requests that are transported by the transaction layer to the I/O devices using a packet-based, split-transaction protocol. The link layer adds sequence numbers and CRC to these packets to create a highly reliable data transfer mechanism. The basic physical layer consists of a dual-simplex channel that is implemented as a transmit pair and a receive pair. The initial speed of 2.5 Giga transfers/sec/direction provides a 200MB/s communications channel that is close to twice the classic PCI data rate.

This thesis is organized as follows. chapter 2 gives the overview of PCI–Express Architecture ,chapter 3 explains the functionality of Transaction Layer of PCIe ,chapter 4 describes different verification strategies Chapter 5 describes the Implementation .chapter 6 gives conclusion and presents a summary of work done and future work .

Chapter 2
Literature Review

The processor communicates with other peripherals in the PC through a path of data called bus. Since the release of the first PC, in 1981, up to the present day, several types of bus have been developed in order to allow the communication between the processor and input and output peripherals. We can name the following buses already launched:

- ISA
- EISA
- MCA
- VLB
- PCI
- AGP
- PCI Express

The main difference among the several types of bus is in the number of bits that can be transmitted at a time, and in the operating frequency used. Nowadays the two fastest types of PC expansion bus are the PCI and the AGP. The PCI-X bus is an extension of the PCI bus designed to the market of network servers.

The Original PCI Bus

The original concept of PCI was chip-to-chip interconnection developed by Intel Corporation. Indeed, PCI is an acronym for *peripheral component interconnects.* Prior to the public release of the PCI bus specification, it was modified to include add-in card slots. With the addition of add-in card slots Intel made a decision in the early 1990s to focus on PCI, to depart from the bus standards of their Multi Bus I and II buses, and the other industrial bus standard VESA. As the PCI bus protocol was included in platforms it eventually replaced the ISA and EISA buses in personal computers and servers. The implementation of the PCI bus in both high-end personal computers and servers placed demands on the PCI bus to provide higher levels of

performance; consequently, the PCI bus evolved from 32 to 64 data bits and its clock reference from 33.3 megahertz to 66.6 megahertz.

2.1 PCI Architecture

The PCI bus was released by Intel in June, 1992. Since then, almost all PC expansion peripherals, such as hard disks, sound cards, LAN cards, and video cards have been using the PCI bus. **PCI Protocol was a Parallel Protocol.**

PCI Platform Architecture and Performance

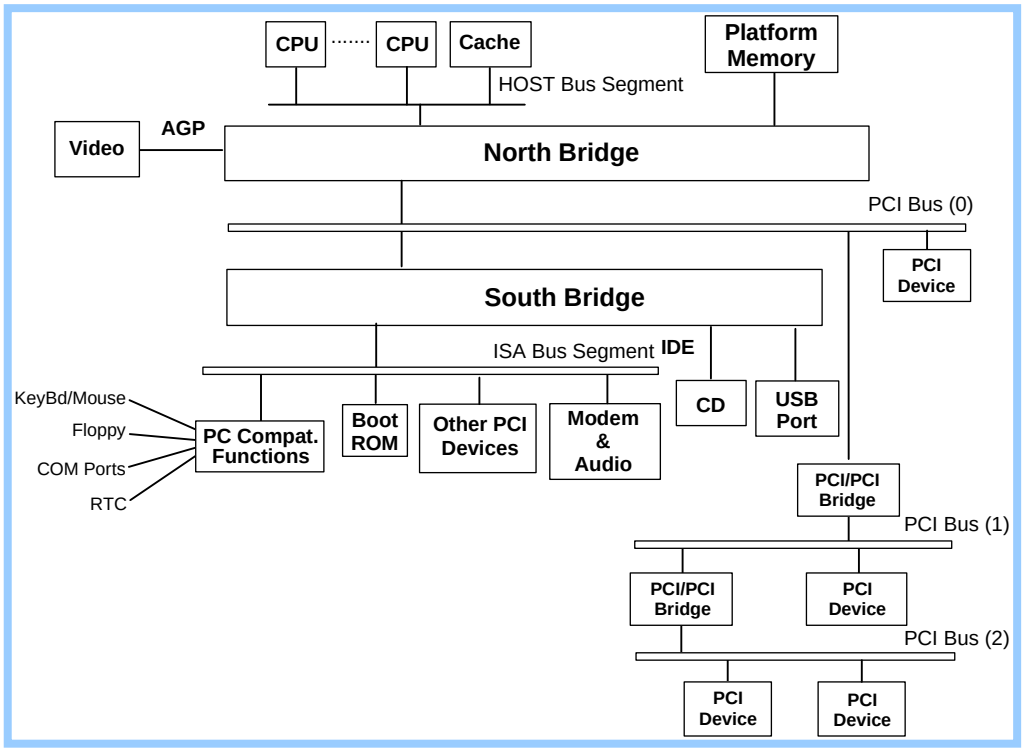


Figure 2-1 33 MHz PCI Bus Based Platform

- PCI platforms evolved into North and South bridges represented by Memory and I/O Controller Hubs.
- Primary concept is that high performance resources are connected to the Memory Controller Hub and the lower performance resources are connected to the I/O Controller Hub.

- Within the lower performance resources of the I/O Controller Hub, some require a high performance connection (e.g. CD) and others can used a shared PCI bus segment (e.g. Ethernet and SCSI)

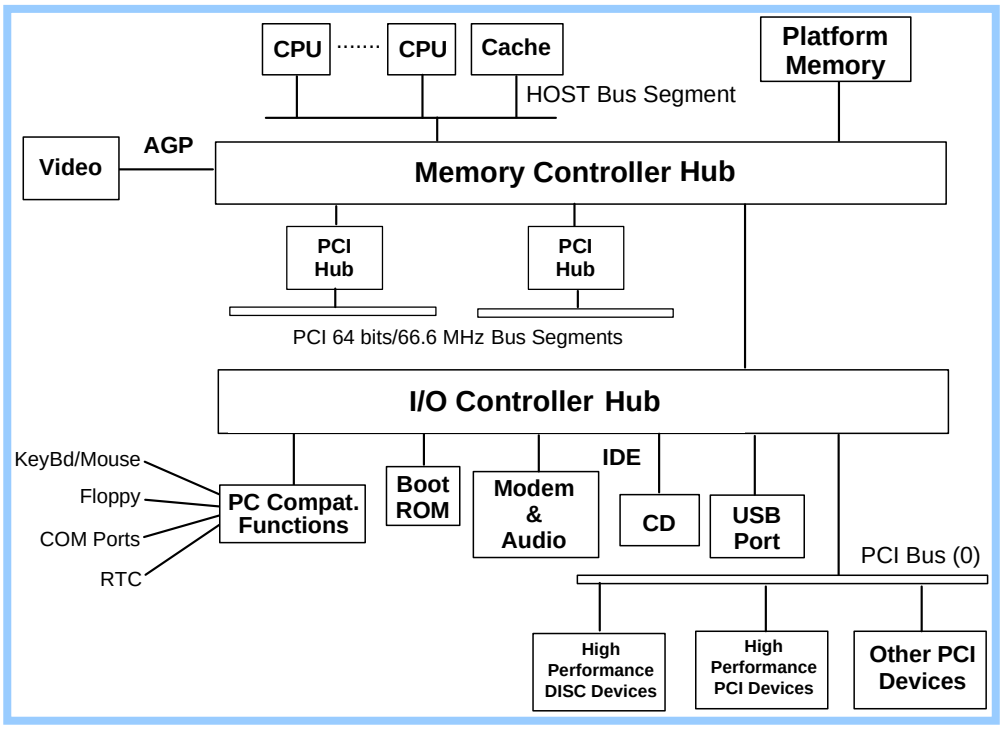


Fig 2-2 Hub Architecture with 66.6 MHz Bus Segments

PCI is designed to support a variety of devices residing on each bus segment.

This **design approach has several advantages** for desktop systems such as:

- Traces on the motherboard reduced to a single set because of shared buses.
- A single protocol (PCI) used for all embedded devices, add-in cards, and chipset components connected to the bus segment.
- As technology evolved there was a greater need to support more high performance resources. The I/O Controller Hub did not provide sufficient performance for all high performance resources relative to Platform Memory. Additional high performance PCI segments were added to the Memory Controller
- In addition to the connection to the Memory Controller Hub, the PCI bus segment size and frequency was increased.
- At 64 data bits PCI achieves bandwidths of 532.8 Megabytes/second.

Table 2-1 lists the data bus widths, number of signal lines, reference clock frequencies, maximum bandwidth, and the maximum number of connectors supported by PCI.

Data Bus Width	Signal Lines (excluding power, arbitration, and test)	Clock Signal Line Frequency (MHz)	Maximum Bandwidth (Megabytes/ second)	Maximum Add-in Card Slots
PCI 32 bits	49	33.3	133.2	4 (1)
PCI 32 bits	49	66.6	266.4	2 (1)
PCI 64 bits	81	33.3	266.4	2 (1)
PCI 64 bits	81	66.6	532.8	2 (1)

Table 2-1 PCI Bus Bandwidth and Add-in Card Slot Limitation

It is also possible to integrate the two Hubs into a single Host/PCI Bridge. The balance of this tutorial will assume a single HOST/PCI or HOST/PCI-X Bridge.

2.2 PCI-X Architecture

- PCI-X was developed to extend performance beyond PCI.
- The HOST/PCI-X Bridge in this discussion represents a consolidation of the Hub controllers into a single bridge structure.
- PCI-X initially simply increased the CLK signal line frequency over PCI to increase bus segment bandwidth. Eventually, PCI-X DDR and QDR provided source synchronous strobes to improve bus segment bandwidth. “D” and “Q” refers to the two and four strobe points within a single CLK signal line period, respectively. Under future consideration is PCI-X 3.0 with source synchronous.
- The increase in bus segment bandwidth greatly reduces the number of add-in card slot per bus segment.

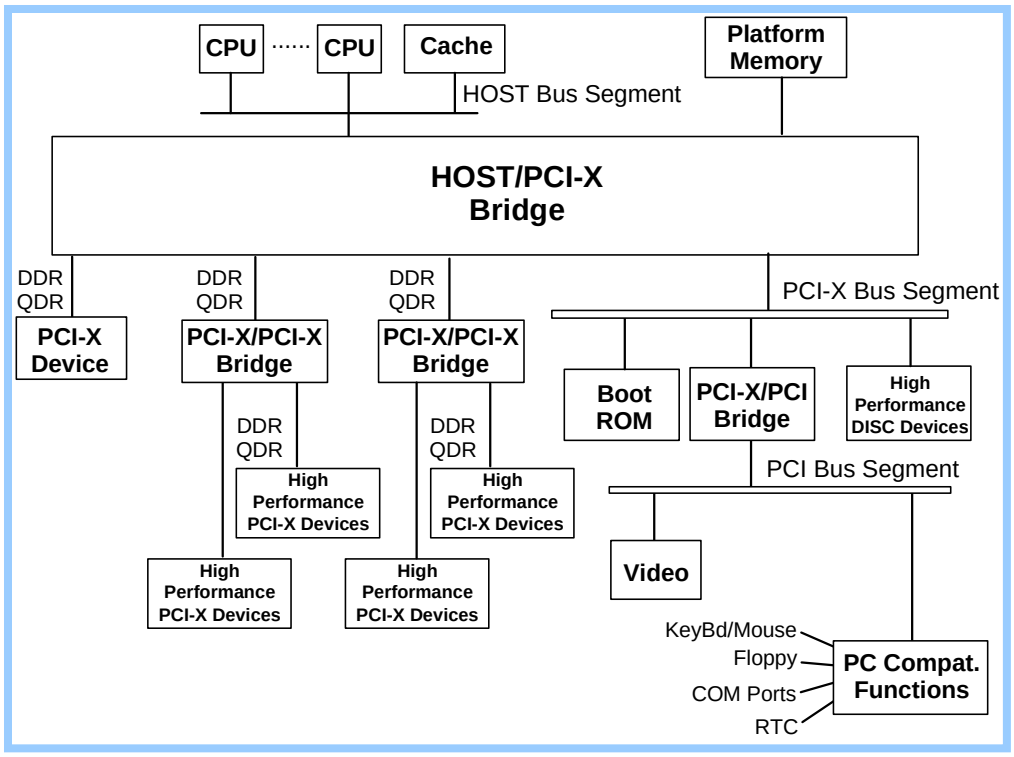


Fig 2-3 Overview of PCI-X DDR/QDR Platform: Highest Performance.

- The resulting higher performance PCI-X DDR and QDR results in point-to-point interconnections. At 64 data bits PCI-X achieves bandwidths over 8524.8 Megabytes/second. This a 16 times improvement over PCI.
- The large signal line count defined for a bus segment shared among several PCI-X devices is impractical for point-to-point interconnections.

The thing is, the PCI bus maximum transfer rate - 133 MB/s – proved to be insufficient for modern 3D applications and it represented a limitation to the development of more sophisticated video cards. In order to solve that issue, Intel created a new bus, called AGP, to increase the transfer rate of video cards – now they wouldn't have to be installed in the PCI bus anymore, but in the AGP bus, which is faster. Then the PCI was not so "busy" anymore, since video cards were the great responsible for the intense traffic in the PCI bus. Even AGP could not solve the Problem as it needs so many pins which was so robust in the mother boards . so finally in March 2002 Intel came out with a solution named PCI-Express highly scalable high Performance third generation I/O Interconnect.

2.3 PCI-Express Architecture

PCI Express provides a high-speed, high-performance, point-to-point, dual simplex, differential signaling Link for interconnecting devices. Data is transmitted from a device on one set of signals, and received on another set of signals.

PCI Express Link

A Link represents a dual-simplex communications channel between two components. The fundamental PCI Express Link consists of two, low-voltage, differentially driven signal pairs: a Transmit pair and a Receive pair as shown in Figure 2-4.

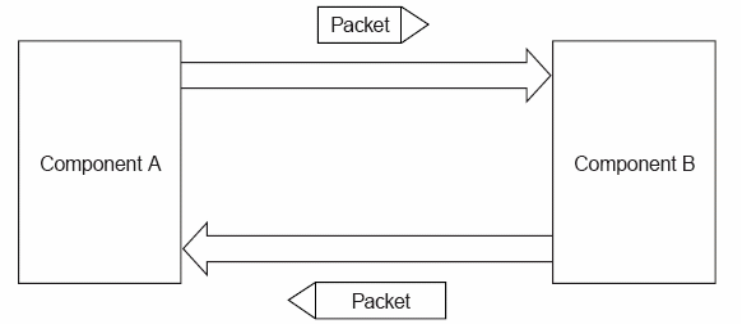


Fig 2-4 PCI-Express Link

Differential Signaling

PCI Express devices employ differential drivers and receivers at each port. This uses differential electrical characteristics of a PCI Express signal. A positive voltage difference between the D+ and D- terminals implies Logical 1. A negative voltage difference between D+ and D- implies a Logical 0. No voltage difference between D+ and D- means that the driver is in the high-impedance tri-state condition, which is referred to as the electrical-idle and low-power state of the Link.

The PCI Express Differential Peak-to-Peak signal voltage at the transmitter ranges from 800 mV - 1200 mV, while the differential peak voltage is one-half these values. The common mode voltage can be any voltage between 0 V and 3.6 V. The differential driver is DC isolated from the differential receiver at the opposite end of the Link by placing a

capacitor at the driver side of the Link. Two devices at opposite ends of a Link may support different DC common mode voltages. The differential impedance at the receiver is matched with the board impedance to prevent reflections from occurring.

2.3.1 PCI Express Fabric Topology

A fabric is composed of point-to-point Links that interconnect a set of components – an example fabric topology is shown in Figure 1-2. This figure illustrates a single fabric instance referred to as a hierarchy – composed of a Root Complex (RC), multiple Endpoints (I/O devices), a Switch, and a PCI Express-PCI Bridge, all interconnected via PCI Express Links.

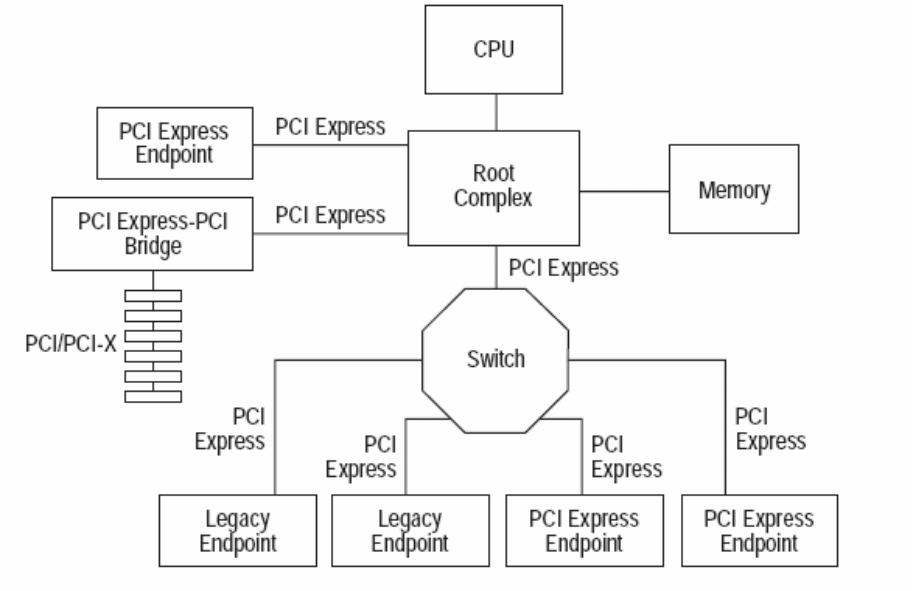


Fig 2-5 PCI-Express Topology

Before going into further discussion of PCI-Express must know some acronyms

A **Requester** is a device that originates a transaction in the PCI Express fabric. Root complex and endpoints are requester type devices.

A **Completer** is a device addressed or targeted by a requester. A requester reads data from a completer or writes data to a completer. Root complex and endpoints are completer type devices.

A **Port** is the interface between a PCI Express component and the Link. It consists of differential transmitters and receivers.

An **Upstream** Port is a port that points in the direction of the root complex.

A **Downstream** Port is a port that points away from the root complex.

An **endpoint** port is an upstream port. A root complex port(s) is a downstream port.

An **Ingress** Port is a port that receives a packet.

An **Egress** Port is a port that transmits a packet.

A **Root Complex (RC)** denotes the root of an I/O hierarchy that connects the CPU/memory subsystem to the I/O. As illustrated in Figure 2-5, a Root Complex may support one or more PCI Express Ports. Each interface defines a separate hierarchy domain. Each hierarchy domain may be composed of a single Endpoint or a sub-hierarchy containing one or more Switch components and Endpoints.

A **Endpoint** refers to a type of device that can be the Requester or Completer of a PCI Express transaction either on its own behalf or on behalf of a distinct non-PCI Express device (other than a PCI device or Host CPU), e.g., a PCI Express attached graphics controller or a PCI Express-USB host controller. Endpoints are classified as either legacy, PCI Express, or Root Complex Integrated Endpoints.

A PCI Express **Endpoint** must be a device with a Type 00h Configuration Space header. It must support Configuration Requests as a Completer, must not depend on operating system allocation of I/O resources claimed through BAR(s).

A PCI Express **Endpoint** must not generate I/O Requests, must not support Locked Requests as a Completer or generate them as a Requestor. PCI Express-compliant software drivers and applications must be written to prevent the use of lock semantics when accessing a PCI Express Endpoint. A PCI Express Endpoint operating as the Requester of a Memory Transaction is required to be capable of generating addresses greater than 4 GB.

Switch

A Switch is defined as a logical assembly of multiple virtual PCI-to-PCI Bridge devices as illustrated in Figure 1-3. All Switches are governed by the following base rules.

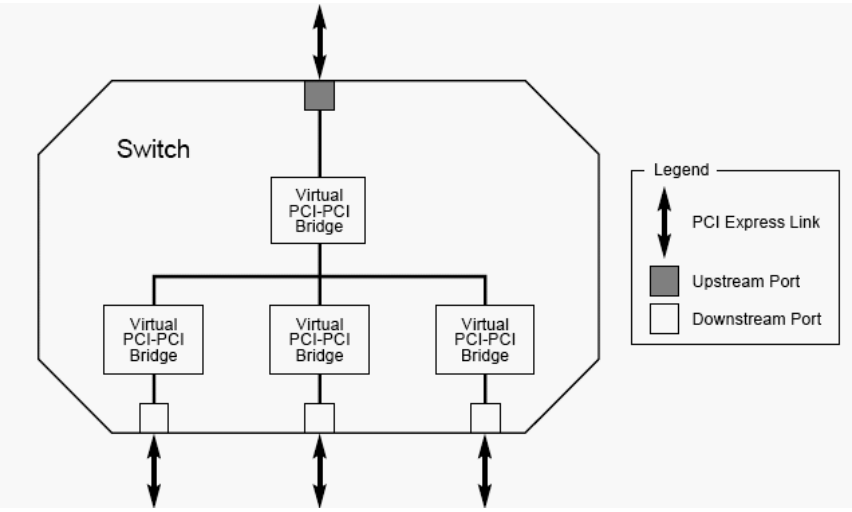


Fig 2-6 PCI – Express Switch

Switches are implemented in systems requiring multiple devices to be interconnected. Switches can range from a 2-port device to an n-port device, where each port connects to a PCI Express Link. A switch may be incorporated into a Root Complex device (Host bridge or North bridge equivalent), resulting in a multi-port root complex.

Switches appear to configuration software as two or more logical PCI-to-PCI Bridges.

- A Switch forwards transactions using PCI Bridge mechanisms; e.g., address based routing.
- Except as noted in this document, a Switch must forward all types of Transaction Layer Packets between any set of Ports.
- Locked Requests must be supported as specified in Section 6.5. Switches are not required to support Downstream Ports as initiating Ports for Locked requests.
- Each enabled Switch Port must comply with the flow control specification specified in PCI-Express base specification.
- A Switch is not allowed to split a packet into smaller packets, e.g., a single packet with a 256-byte payload must not be divided into two packets of 128 bytes payload each.
- Arbitration between Ingress Ports (inbound Link) of a Switch may be implemented using round robin or weighted round robin when contention occurs

on the same Virtual Channel. This is described in more detail later within the specification.

- Endpoint devices (represented by Type 00h Configuration Space headers) must not appear to configuration software on the switch's internal bus as peers of the virtual PCI-to-PCI Bridges representing the Switch Downstream Ports.

PCI Express-PCI Bridge

A PCI Express-PCI Bridge provides a connection between a PCI Express fabric and a PCI/PCI-X hierarchy.

Bus	Clock	Number of bits	Data per Clock Cycle	Maximum Transfer Rate
PCI	33 MHz	32	1	133 MB/s
PCI	66 MHz	32	1	266 MB/s
PCI	33 MHz	64	1	266 MB/s
PCI	66 MHz	64	1	533 MB/s
PCI-X 64	66 MHz	64	1	533 MB/s
PCI-X 133	133 MHz	64	1	1.066 MB/s
PCI-X 266	133 MHz	64	2	2.132 MB/s
PCI-X 533	133 MHz	64	4	4.266 MB/s
AGP x1	66 MHz	32	1	266 MB/s
AGP x2	66 MHz	32	2	533 MB/s
AGP x4	66 MHz	32	4	1.066 MB/s
AGP x8	66 MHz	32	8	2.133 MB/s
PCIe	250Mhz	2	64	2.5 Gbits/sec

Table 2-2 Comparison of different Buses

Chapter 3
PCI Express Layering Overview

Overview

The PCI Express has a layered architecture for device. The layers consist of a Transaction Layer, a Data Link Layer and a Physical layer. The layers can be further divided vertically into two, a transmit portion that processes outbound traffic and a receive portion that processes inbound traffic. However, a device design does not have to implement a layered architecture as long as the functionality required by the specification is supported.

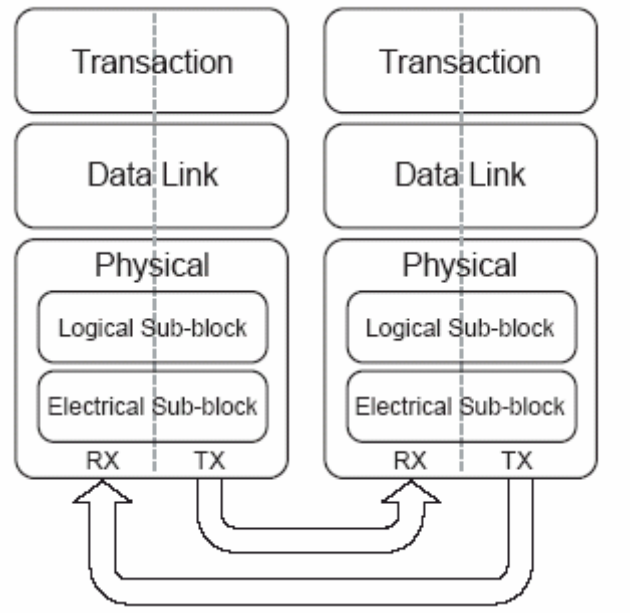


Fig 3-1 PCI-Express Layered structure

The goal of this section is to describe the function of each layer and to describe the flow of events to accomplish a data transfer. Packet creation at a transmitting device and packet reception and decoding at a receiving device are also explained.

Transmit Portion of Device Layers.

Consider the transmit portion of a device. Packet contents are formed in the Transaction Layer with information obtained from the device core and application. The packet is stored in buffers ready for transmission to the lower layers. This packet is referred to as a Transaction Layer Packet (TLP), The Data Link Layer concatenates to the

packet additional information required for error checking at a receiver device. The packet is then encoded in the Physical layer and transmitted differentially on the Link by the analog portion of this Layer. The packet is transmitted using the available Lanes of the Link to the receiving device which is its neighbor.

Receive Portion of Device Layers

The receiver device decodes the incoming packet contents in the Physical Layer and forwards the resulting contents to the upper layers. The Data Link Layer checks for errors in the incoming packet and if there are no errors forwards the packet up to the Transaction Layer. The Transaction Layer buffers the incoming TLPs and converts the information in the packet to a representation that can be processed by the device core and application.

Device Layers and their Associated Packets

Three categories of packets are defined, each one is associated with one of the three device layers. Associated with the Transaction Layer is the Transaction Layer Packet (TLP). Associated with the Data Link Layer is the Data Link Layer Packet (DLLP). Associated with the Physical Layer is the Physical Layer Packet (PLP). These packets are introduced next.

3.1 Transaction Layer

The upper Layer of the architecture is the Transaction Layer. The Transaction Layer's primary responsibility is the assembly and disassembly of Transaction Layer Packets (TLPs). TLPs are used to communicate transactions, such as read and write, as well as certain types of events. The Transaction Layer is also responsible for managing credit-based flow control for TLPs. Every request packet requiring a response packet is implemented as a split transaction. Each packet has a unique identifier that enables response packets to be directed to the correct originator. The packet format supports different forms of addressing depending on the type of the transaction (Memory, I/O, Configuration, and Message). The Packets may also have attributes such as No Snoop and Relaxed Ordering. The transaction Layer supports four address spaces: it includes the three PCI address spaces (memory, I/O, and configuration)

and adds a Message Space. This specification uses Message Space to support all prior sideband signals, such as interrupts, power-management requests, and so on, as in-band Message transactions. You could think of PCI Express Message transactions as "virtual wires" since their effect is to eliminate the wide array of sideband signals currently used in a platform implementation.

3.1.1 Transaction Layer Packets (TLPs)

PCI Express transactions employ TLPs which originate at the Transaction Layer of a transmitter device and terminate at the Transaction Layer of a receiver device. This process is represented in **Fig 2-8** . The Data Link Layer and Physical Layer also contribute to TLP assembly as the TLP moves through the layers of the transmitting device. At the other end of the Link where a neighbor receives the TLP, the Physical Layer, Data Link Layer and Transaction Layer disassemble the TLP

TLP Packet Assembly

A TLP that is transmitted on the Link appears as shown below.

.

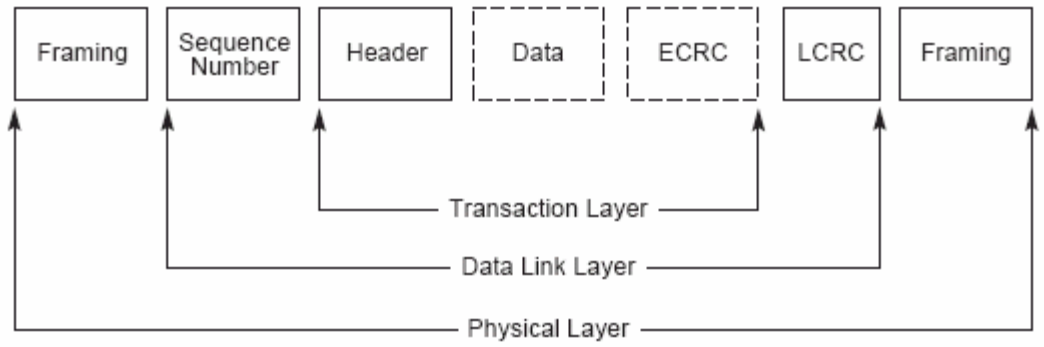


Fig 3-2 TLP operation at different layers

The software layer/device core sends to the Transaction Layer the information required to assemble the core section of the TLP which is the header and data portion of the packet. Some TLPs do not contain a data section. An optional End-to-End CRC (ECRC) field is calculated and appended to the packet. The ECRC field is used by the ultimate targeted device of this packet to check for CRC errors in the header and data portion of the TLP.

TLP Packet Disassembly

A neighboring receiver device receives the incoming TLP bit stream. As shown in **Fig 2-8** the received TLP is decoded by the Physical Layer and the Start and End frame fields are stripped. The resultant TLP is sent to the Data Link Layer. This layer checks for any errors in the TLP and strips the sequence ID and LCRC field. Assume there are no LCRC errors, and then the TLP is forwarded up to the Transaction Layer. If the receiving device is a switch, then the packet is routed from one port of the switch to an egress port based on address information contained in the header portion of the TLP. Switches are allowed to check for ECRC errors and even report the errors it finds and error. However, a switch is not allowed to modify the ECRC that way the targeted device of this TLP will detect an ECRC error if there is such an error.

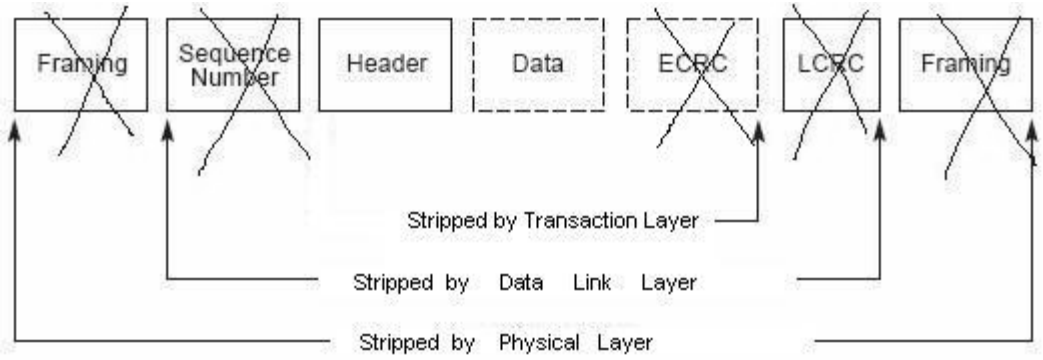


Fig 3-3 Packet disassembly operation

3.2 Data Link Layer

The middle Layer in the stack, the Data Link Layer, serves as an intermediate stage between the Transaction Layer and the Physical Layer. The primary responsibilities of the Data Link Layer include Link management and data integrity, including error detection and error correction. The transmission side of the Data Link Layer accepts TLPs assembled by the Transaction Layer, calculates and applies a data protection code and TLP sequence number, and submits them to Physical Layer for transmission across the Link. The receiving Data Link Layer is responsible for checking the integrity of received TLPs and for submitting them to the Transaction Layer for further processing. On detection of TLP error(s), this Layer is responsible for requesting

retransmission of TLPs until information is correctly received, or the Link is determined to have failed.

3.2.1 Data Link Layer Services

The Data Link Layer is responsible for reliably exchanging information with its counterpart on the opposite side of the Link.

- Initialization and power management services:
- Accept power state Requests from the Transaction Layer and convey to the Physical Layer
- Convey active/reset/disconnected/power managed state to the Transaction Layer
- Data protection, error checking, and retry services:
- CRC generation
- Transmitted TLP storage for Data Link level retry
- Error checking
- TLP acknowledgment and retry Messages
- Error indication for error reporting and logging

3.3 Physical Layer

The Physical Layer includes all circuitry for interface operation, including driver and input buffers, parallel-to-serial and serial-to-parallel conversion, PLL(s), and impedance matching circuitry. It includes also logical functions related to interface initialization and maintenance. The Physical Layer exchanges information with the Data Link Layer in an implementation-specific format. This Layer is responsible for converting information received from the Data Link Layer into an appropriate serialized format and transmitting it across the PCI Express Link at a frequency and width

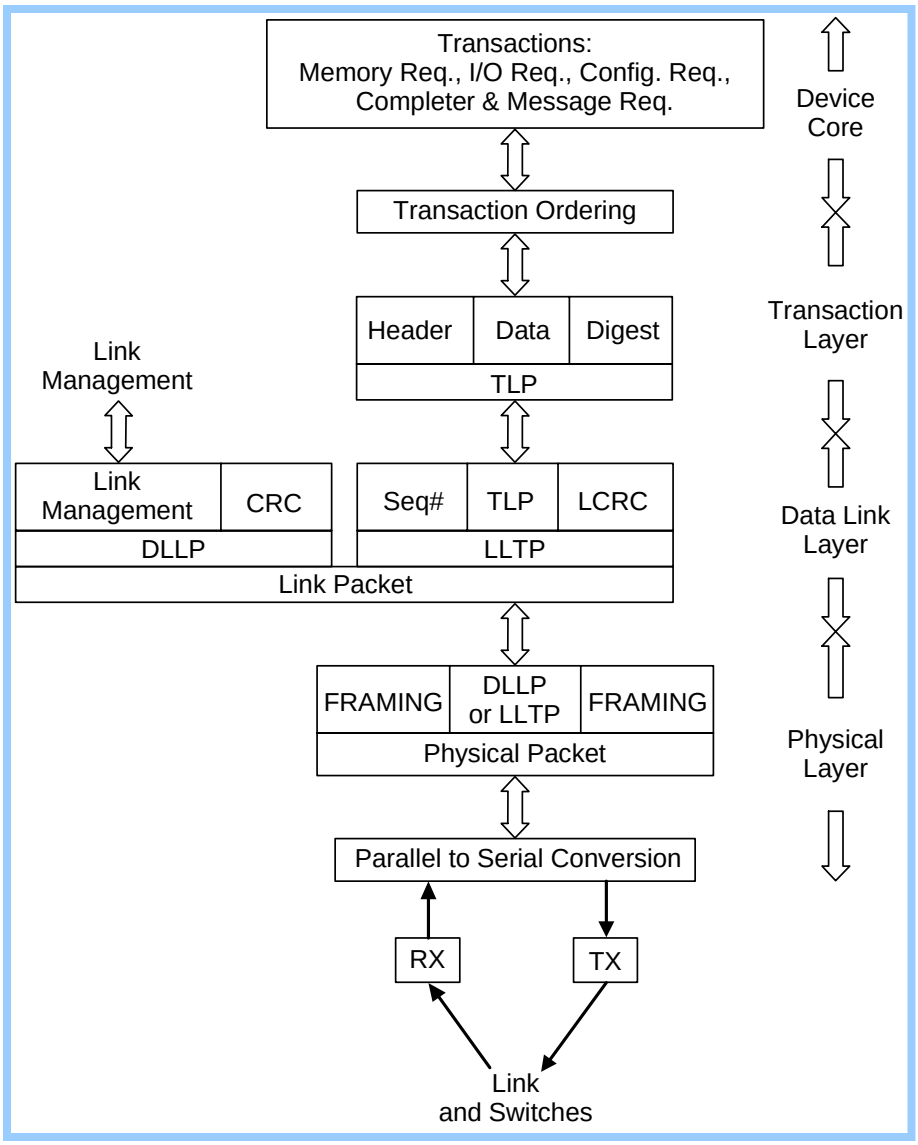


Fig 3-4 Packet Flow from Transaction layer to Link

compatible with the device connected to the other side of the Link. The PCI Express architecture has “hooks” to support future performance enhancements via speed upgrades and advanced encoding techniques. The future speeds, encoding techniques or media may only impact the Physical Layer definition.

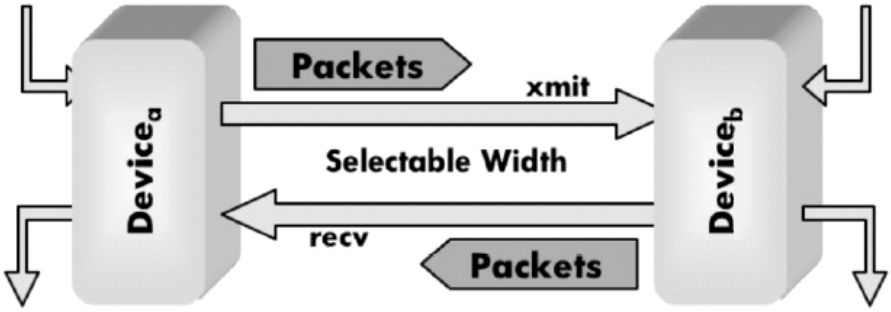


Fig 3-5 Link Between two devices

3.3.1 Physical Layer Services

Interface initialization, maintenance control, and status tracking:

- Reset/Hot-Plug control/status
- Interconnect power management
- Width and Lane mapping negotiation
- Polarity reversal

Symbol and special ordered set generation:

- 8-bit/10-bit encoding/decoding
- Embedded clock tuning and alignment

Symbol transmission and alignment:

- Transmission circuits
- Reception circuits
- Elastic buffer at receiving side
- Multi-Lane de-skew (for widths > x1) at receiving side
- System DFT support features

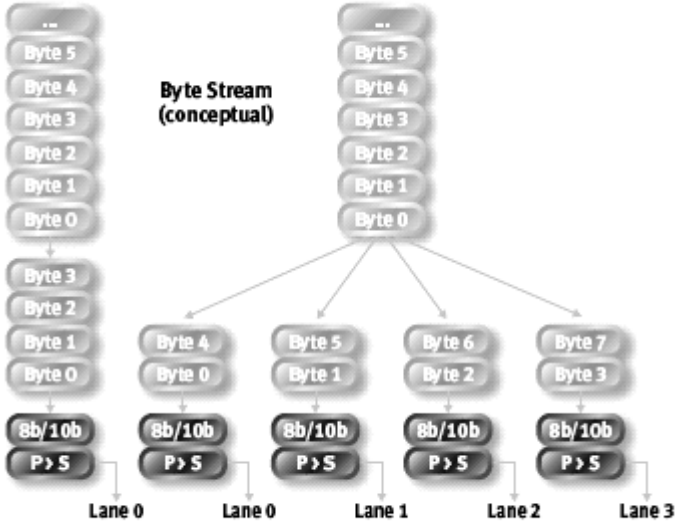


Fig 3-6 Serial Bit stream transmission on to different lanes

The bandwidth of a PCI Express link may be linearly scaled by adding signal pairs to form multiple lanes. The physical layer supports x1, x2, x4, x8, x12, x16 and x32 lane widths and splits the byte data as shown in Figure 9. Each byte is transmitted, with 8b/10b encoding, across the lane(s). This data disassembly and reassembly is transparent to other layers. During initialization, each PCI Express link is set up following a negotiation of lane widths and frequency of operation by the two agents at each end of the link. No firmware or operating system software is involved. The PCI Express architecture comprehends future performance enhancements via speed upgrades and advanced encoding techniques. The future speeds, encoding techniques or media would only impact the physical layer.

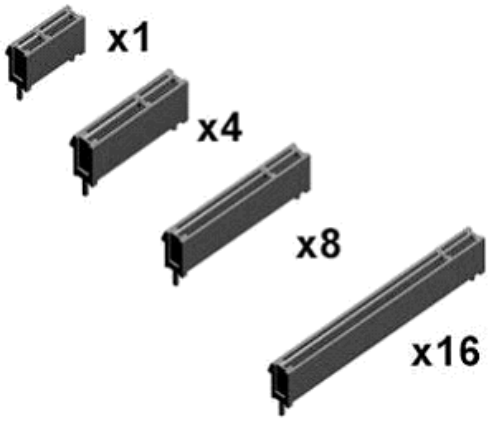


Fig 3-7 PCI-Express Cross Link lanes

3.4 Why serial?

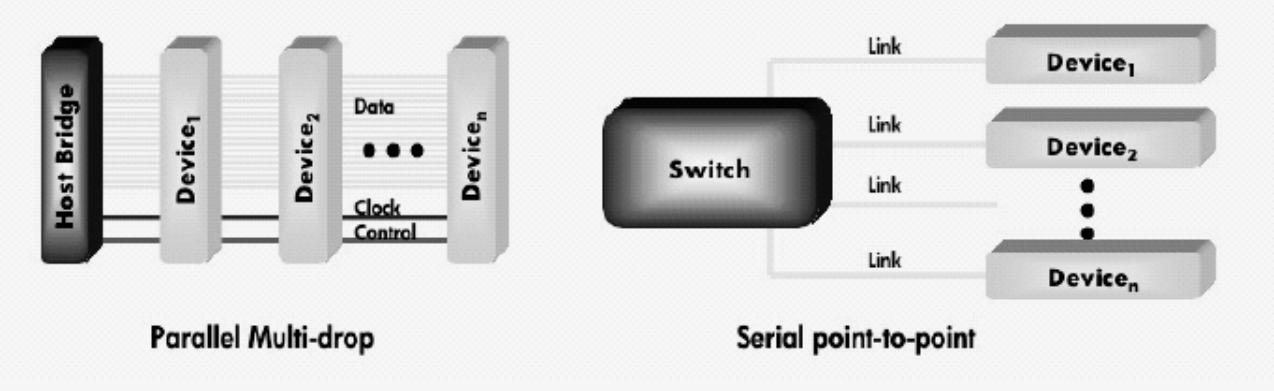


Fig 3-8 Serial Vs Parallel communication

The serial communication differs from the parallel one for only transmitting a bit at a time, while in the parallel communication several bits are transmitted per time. That makes the **parallel communication faster than the serial** one.

That statement, however accepted by most people, is not totally true. The **serial communication may be faster than the parallel one**, **all you need is that the bits leave the transmitting device at a much higher speed.** An example of such is the ATA Serial port that however serial can reach a transfer rate of up to 150 MB/s, while the traditional IDE port gets to reach 133 MB/s at the most.

There are several reasons to make the devices migrate from the parallel communication to the serial one. **In the parallel communication**, since several bits are transmitted per time, **a wire is required per each bit**. For instance, in a 32 bit communication (as it is the case of the PCI slot) 32 wires are required just for the data transmission, not to mention the additional control signals that are necessary. The higher the quantity of bits being transmitted per time, the more wires are used, making the creation of cables and the construction of boards difficult. In the serial communication, only two wires are required, making it much easier to project the communication between two devices.

The higher the transfer rate, the bigger the problem with the electromagnetic interference. Each wire becomes an antenna in potential, capturing a lot of noise from

the environment, which may corrupt the data transmitted. In the parallel communication, since many wires are used, the problem of the electromagnetic interference is a serious one. In the serial communication, on the other hand, since only two wires are used, that problem is much more easily solved, by simply protecting the two wires used.

There is yet another problem, a not much discussed one. Even though we say that in the parallel communication all the bits are transmitted at the same time, the bits do not get to the receiver exactly at the same time. If in low performance devices the small time difference in the reception of the several bits of data is not important, in high-speed devices that difference in the reception time of the several bits makes the device wastes time having to wait for all the bits to arrive, which may represent a significant fall in performance, since the data transmission operation happens in very short times.

Another difference between the parallel communication and the serial one is that the **parallel communication is half-duplex, while the serial one is full-duplex**. In plain English, that means simply the following: in the parallel communication, the only path between the transmitter and the receiver is used both for the transmission and for the reception of the data. Since there is only one path, it is not possible to transmit and receive data at the same time. In the serial communication, on the other hand, since it only uses two wires, the manufacturers usually make four wires available, two for the transmission and two for the reception of data. In other words, a path just for the transmission of data, and another one only for its reception. That makes it possible for the simultaneous transmission and reception of data. Such architectural difference alone makes the serial communication potentially twice as fast as the parallel communication, if we compare two communications that have the same transfer rate.

Chapter 4
Transaction Layer

4.1 Transaction Layer Overview

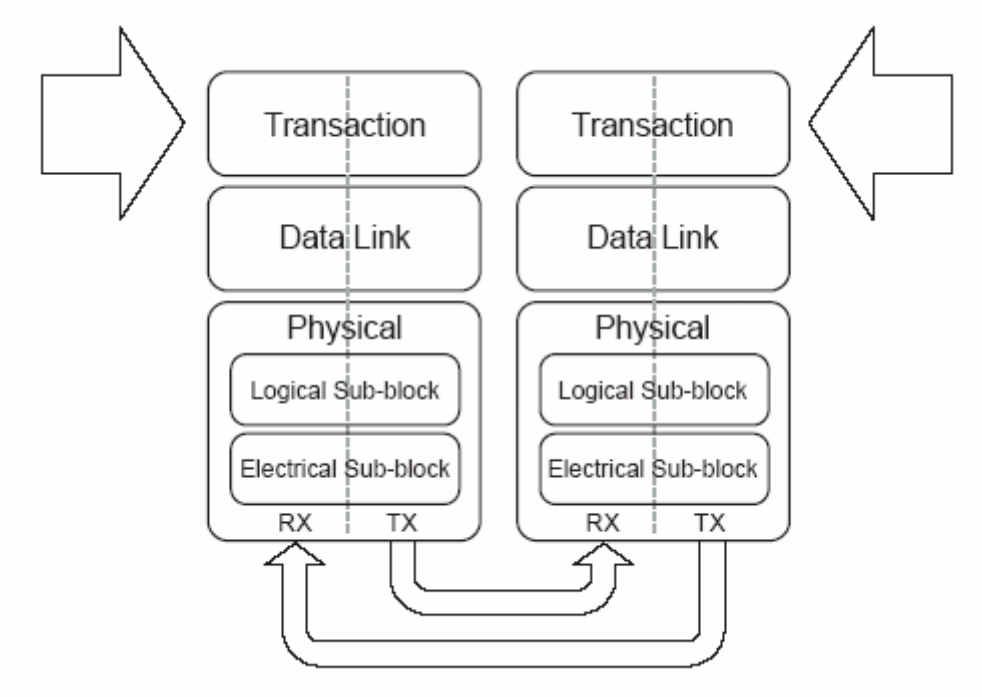


Fig 4-1 Transaction Layer Overview

At a high level, the key aspects of PCI Express associated with the Transaction Layer are:

- A pipelined full split-transaction protocol
- Mechanisms for differentiating the ordering and processing requirements of Transaction Layer Packets (TLPs)
- Credit-based flow control
- Optional support for data poisoning and end-to-end data integrity detection.

The Transaction Layer comprehends the following:

- TLP construction and processing
- Association of PCI Express transaction-level mechanisms with device resources including:
  1. Flow Control
  2. Virtual Channel management
- Rules for ordering and management of TLPs

• PCI/PCI-X compatible ordering

• Including Traffic Class differentiation

This chapter specifies the behaviors associated with the Transaction Layer.

The transaction layer receives read and write requests from the software layer and creates request packets for transmission to the link layer. All requests are implemented as split transactions and some of the request packets require a response packet. The transaction layer also receives response packets from the link layer and matches these with the original software requests. Each packet has a unique identifier that enables response packets to be directed to the correct originator. The packet format offers 32-bit memory addressing and extended 64-bit memory addressing. Packets also have attributes such as "no-snoop," "relaxed ordering," and "priority," which may be used to route these packets optimally through the I/O subsystem.

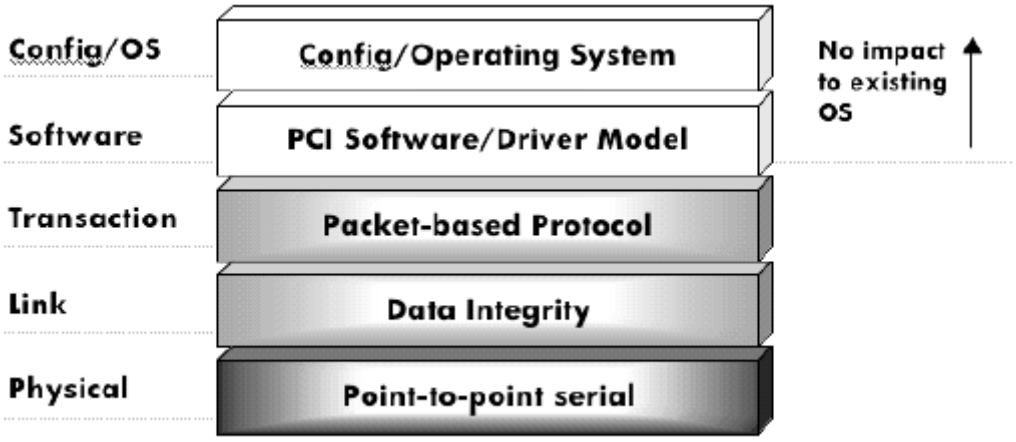


Fig 4-2 software Layers operating on PCI-Express Layers

The transaction layer provides four address spaces – three PCI address spaces (memory, I/O and configuration) and message space. PCI 2.2 introduced an alternate method of propagating system interrupts called message signaled interrupt (MSI). Here a special-format memory-write transaction was used instead of a hard-wired sideband signal, as an optional capability in a PCI 2.2 system. The PCI Express specification reuses the MSI concept as a primary method for interrupt processing and uses a message space to accept all prior sideband signals, such as interrupts, power-management requests, and resets, as

in-band messages. Other "special cycles" within the PCI 2.2 specification, such as interrupt acknowledge, are also implemented as in-band messages. You could think of PCI Express messages as "virtual wires" because their effect is to eliminate the wide array of sideband signals currently used in a platform implementation.

4.1.1. Address Spaces, Transaction Types, and Usage

Transactions form the basis for information transfer between a Requester and Completer. Four address spaces are defined within the PCI Express architecture, and different Transaction types are defined, each with its own unique intended usage, as shown in Table below.

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 an I/O-mapped location
Configuration	Read Write	Device configuration/setup
Message	Baseline (including Vendor–defined)	From event signaling mechanism to general purpose messaging

Table 4-1 Address Spaces and their usage

The different kinds of transactions in PCI-Express are classified as

4.1.1.1. Memory Transactions

Memory Transactions include the following types:

- Read Request/Completion
- Write Request

Memory Transactions use two different address formats:

- Short Address Format: 32-bit address
- Long Address Format: 64-bit address

4.1.1.2. I/O Transactions

PCI Express supports I/O Space for compatability with legacy devices which require their use.Future revisions of this specification are expected to depreciate the use of I/O Space.

I/O Transactions include the following types:

- Read Request/Completion
- Write Request/Completion

I/O Transactions use a single address format:

- Short Address Format: 32-bit address

4.1.1.3. Configuration Transactions

Configuration Transactions are used to access configuration registers of PCI Express devices.

Configuration Transactions include the following types:

- Read Request/Completion
- Write Request/Completion

4.1.1.4. Message Transactions

The Message Transactions, or simply Messages, are used to support in-band communication of events between PCI Express devices.

In addition to the specified Messages, PCI Express provides support for vendor-defined Messages using specified Message codes. This establishes a standard framework within which vendors can specify their own vendor-defined Messages tailored to fit the specific requirements of their platforms.

4.2 Packet Format Overview

Transactions consist of Requests and Completions, which are communicated using packets. Figure 3-2 shows a high level serialized view of a Transaction Layer Packet (TLP), consisting of a TLP header, a data payload (for some types of packets), and an optional TLP digest. Figure 2-3 shows a more detailed view of the TLP. The following sections of this chapter define the detailed structure of the packet headers and digest

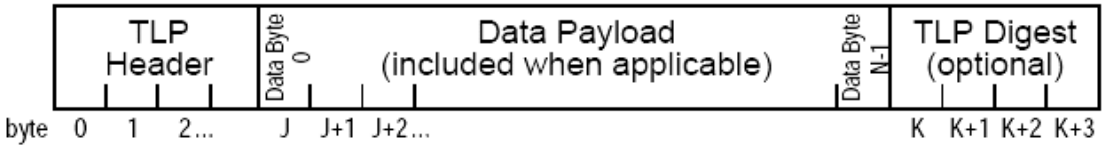


Fig 4-3 PCI-Express Packet Format View

PCI Express conceptually transfers information as a serialized stream of bytes as shown in Figure 3-6. Note that at the byte level, information is transmitted/received over the interconnect with byte 0 being transmitted/received first. For details on how individual bytes of the packet are encoded and transmitted over the physical media. Detailed layouts of the TLP header and TLP digest (presented in generic form in Figure 2-9) are drawn with the lower numbered bytes on the left rather than on the right as has traditionally been depicted in other PCI specifications. The PCI Express header layout is optimized for performance on a serialized interconnect, driven by the requirement that the most time critical information be transferred first. For example, within the PCI Express TLP header, the most significant byte of the address field is transferred first so that it may be used for early address decode.

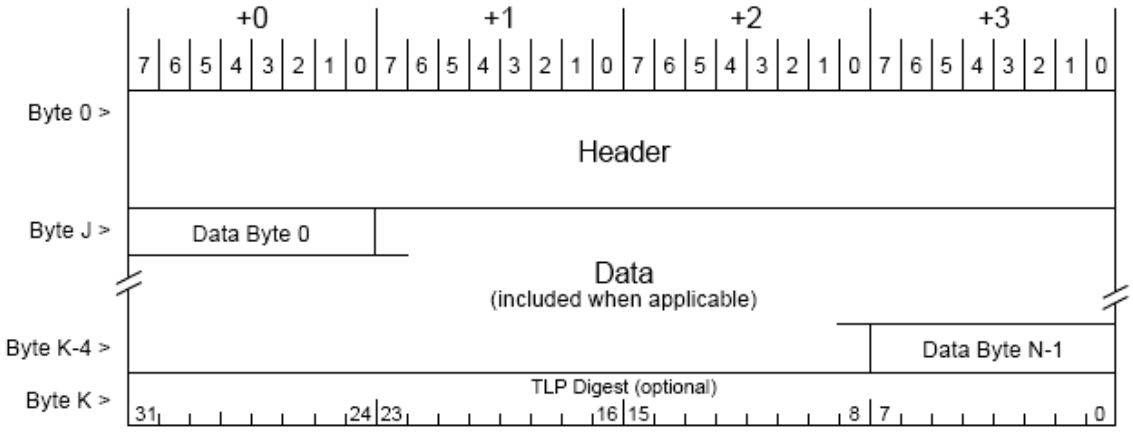


Fig 4-4 PCI Express Packet in detail

Payload data within a TLP is depicted with the lowest addressed byte (byte J in Figure 3-3) shown to the upper left. This retains the traditional PCI byte layout with the lowest

addressed byte shown on the right. Regardless of depiction, all bytes are conceptually transmitted over the Link in increasing byte number order.

Depending on the type of a packet, the header for that packet will include some of the following types of fields:

- Format of the packet
- Type of the packet
- Length for any associated data
- Transaction Descriptor, including:
  - Transaction ID
  - Attributes
  - Traffic Class
- Address/routing information
- Byte Enables
- Message encoding
- Completion status

4.3. Transaction Layer Protocol - Packet

Definition

PCI Express uses a packet based protocol to exchange information between the Transaction Layers of the two components communicating with each other over the Link. PCI Express supports the following basic transaction types: Memory, I/O, Configuration, and Messages. Two addressing formats for Memory Requests are supported: 32 bit and 64 bit.

Transactions are carried using Requests and Completions. Completions are used only where required, for example, to return read data, or to acknowledge Completion of I/O and Configuration Write Transactions. Completions are associated with their corresponding Requests by the value in the Transaction ID field of the Packet header.

All TLP fields marked **Reserved** (sometimes abbreviated as R) must be filled with all 0's when a TLP is formed. Values in such fields must be ignored by Receivers and

forwarded unmodified by Switches. Note that for certain fields there are both specified and reserved values – the handling of reserved values in these cases is specified separately for each case.

4.3.1. Common Packet Header Fields

All Transaction Layer Packet (TLP) headers contain the following fields .

- Fmt[1:0] – Format of TLP (see **Fig 3-3**) – bits 6:5 of byte 0
- Type[4:0] – Type of TLP – bits 4:0 of byte 0

The Fmt and Type fields provide the information required to determine the size of the remaining part of the header, and if the packet contains a data payload following the header.

The Fmt, Type ,TD, and Length fields contain all information necessary to determine the overall size of the TLP itself. The Type field, in addition to defining the type of the TLP also determines how the TLP is routed by a Switch. Different types of TLPs are discussed in more detail in the following sections.

- Permitted Fmt[1:0] and Type[4:0] field values are shown in Table 2-3.

• All other encodings are reserved.

- TC[2:0] – Traffic Class (see fig 3-4) – bits [6:4] of byte 1
- Attr[1:0] – Attributes (see fig 3-4) – bits [5:4] of byte 2
- TD – 1b indicates presence of TLP digest in the form of a single DW at the end of the TLP– bit 7 of byte 2
- EP – indicates the TLP is poisoned (see fig 3-4) – bit 6 of byte 2 10
- Length[9:0] – Length of data payload in DW (see Table 2-4) – bits 1:0 of byte 2 concatenated with bits 7:0 of byte 3

- TLP data must be 4-byte naturally aligned and in increments of 4-byte Double Words (DW).
- Reserved for TLPs that do not contain or refer to data payloads, including Cpl, CplLk, and Messages (except as specified)

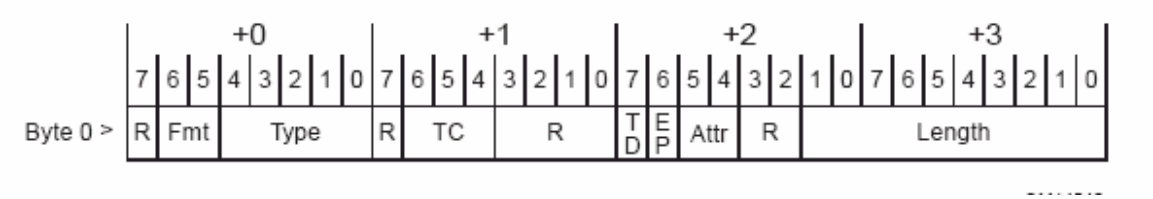


Fig 4-4 Fields Present in all TLPs

Table 4-2 Fmt[1:0] Field Values

Fmt[1:0]	Corresponding TLP Format
00b	3 DW header, no data
01b	4 DW header, no data
10b	3 DW header, with data
11b	4 DW header, with data

The Fmt Type in the header is an identifier which specifies the transaction as classified in the **Table 4-3.**

The Packets at the most hierarchy classified into two categories namely the **TLP WITH DATA PAYLOAD** and **TLP WITH OUT DATA PAYLOAD**

These Packets are further classified into three categories

- **Non-Posted Requests –**
  - MRD (Memory Read Request)
  - I/O-RD (I/O Read Request)
  - I/O-WR(I/O Write Request)
- **Posted Requests --**
  - MWR (Memory Write Request)
- **Completion Type Requests** –
  - CPL(Completion with out data)
  - CPLD(Completion With Data)

Table 4-3 : Fmt[1:0] and Type[4:0] Field Encodings

TLP Type	Fmt [1:0]^2	Type [4:0]	Description
MRd	00 01	0 0000	Memory Read Request
MRdLk	00 01	0 0001	Memory Read Request-Locked
MWr	10 11	0 0000	Memory Write Request
IORd	00	0 0010	I/O Read Request
IOWr	10	0 0010	I/O Write Request
CfgRd0	00	0 0100	Configuration Read Type 0
CfgWr0	10	0 0100	Configuration Write Type 0
CfgRd1	00	0 0101	Configuration Read Type 1
CfgWr1	10	0 0101	Configuration Write Type 1
Msg	01	1 0$r_2r_1r_0$	Message Request – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-11).
MsgD	11	1 0$r_2r_1r_0$	Message Request with data payload – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-11).
Cpl	00	0 1010	Completion without Data – Used for I/O and Configuration Write Completions and Read Completions (I/O, Configuration, or Memory) with Completion Status other than Successful Completion.
CplD	10	0 1010	Completion with Data – Used for Memory, I/O, and Configuration Read Completions.
CplLk	00	0 1011	Completion for Locked Memory Read without Data – Used only in error case.
CplDLk	10	0 1011	Completion for Locked Memory Read – otherwise like CplD.
			All encodings not shown above are Reserved.

Table 4-4: Length[9:0] Field Encoding

Length[9:0]	Corresponding TLP Data Payload Size
00 0000 0001b	1 DW
00 0000 0010b	2 DW
...	...
11 1111 1111b	1023 DW
00 0000 0000b	1024 DW

4.3.2 TLPs with Data Payloads - Rules

- Length is specified as an integral number of DW
- Length[9:0] is reserved for all Messages except those which explicitly refer to a Data Length .
- The Transmitter of a TLP with a data payload must not allow the data payload length as given by the TLP's Length [ ] field to exceed the length specified by the value in the Max_Payload_Size field of the Transmitter's Device Control register taken as an integral number of DW .
- For an Upstream Port associated with a multi-function device whose Max_Payload_Size settings are identical across all functions, a transmitted TLP's data payload must not exceed the common Max_Payload_Size setting.
- For an Upstream Port associated with a multi-function device whose Max_Payload_Size
- settings are not identical across all functions, a transmitted TLP's data payload must not exceed a Max_Payload_Size setting whose determination is implementation specific.
- Transmitter implementations are encouraged to use the Max_Payload_Size setting from the function that generated the transaction, or else the smallest Max_Payload_Size setting across all functions.
- Software should not set the Max_Payload_Size in different functions to different values unless software is aware of the specific implementation.

4.3.3 TLP Digest Rules

- For any TLP, a value of 1b in the TD field indicates the presence of the TLP Digest field including an ECRC value at the end of the TLP
- A TLP where the TD field value does not correspond with the observed size (accounting for the data payload, if present) is a Malformed TLP
  This is a reported error associated with the Receiving Port.

4.3.4. Routing and Addressing Rules

There are three principal mechanisms for TLP routing: address, ID, and implicit. This section defines the rules for the address and ID routing mechanisms. Implicit routing is used only with Message Requests, and is covered in later discussions.

4.3.4.1. Address Based Routing Rules

- Address routing is used with Memory and I/O Requests.
- Two address formats are specified, a 64-bit format used with a 4 DW header and a 32-bit format used with a 3 DW header

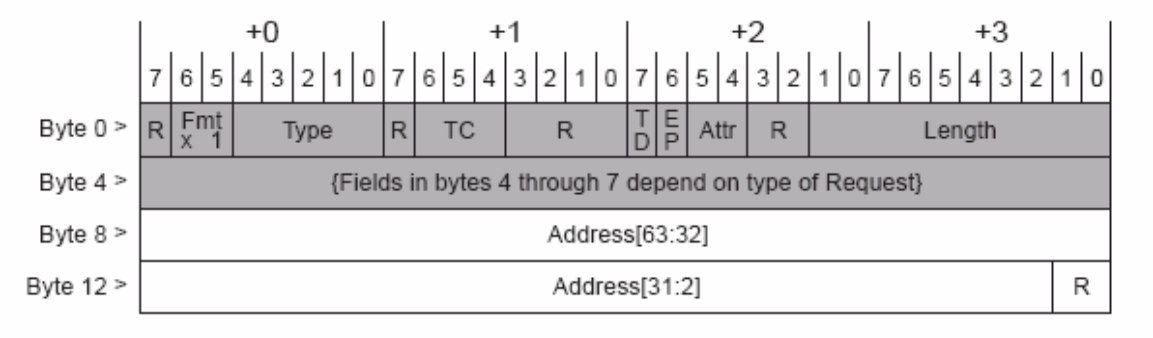


Fig 4-5 64-bit Address Routing

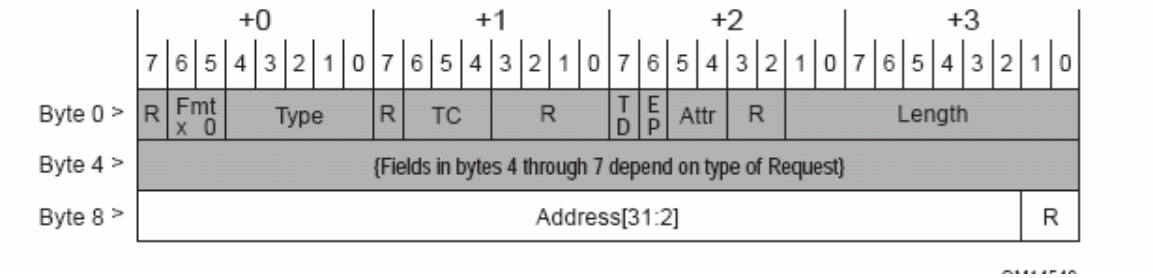


Fig 4-6 32-Bit Address Routing

Address mapping to the TLP header is shown in Table 4-5.

Address Bits	32-bit Addressing	64-bit Addressing
63:56	Not Applicable	Bits 7:0 of Byte 8
55:47	Not Applicable	Bits 7:0 of Byte 9
47:40	Not Applicable	Bits 7:0 of Byte 10
39:32	Not Applicable	Bits 7:0 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

Table 4-5 Address Field Mapping

- Memory Read Requests and Memory Write Requests can use either format.
- For Addresses below 4 GB, Requesters must use the 32-bit format.
- I/O Read Requests and I/O Write Requests use the 32-bit format.
- All PCI Express Agents must decode all address bits in the header - address aliasing is not allowed.

4.3.4.2 ID Based Routing Rules

- ID routing is used with Configuration Requests, optionally with Vendor_Defined Messages, and with Completions
- ID routing uses the Bus, Device, and Function Numbers to specify the destination Device for the TLP
- Bus, Device, and Function Number to TLP header mapping is shown in Table 4-6
- Two ID routing formats are specified, one used with a 4 DW header and one used with a 3 DW header
- Header field locations are the same for both formats, and are given in Figure 4-7.

Field	Header Location
Bus Number[7:0]	Bits 7:0 of Byte 8
Device Number[4:0]	Bits 7:3 of Byte 9
Function Number[2:0]	Bits 2:0 of Byte 9

Table 4-6 Transaction Id

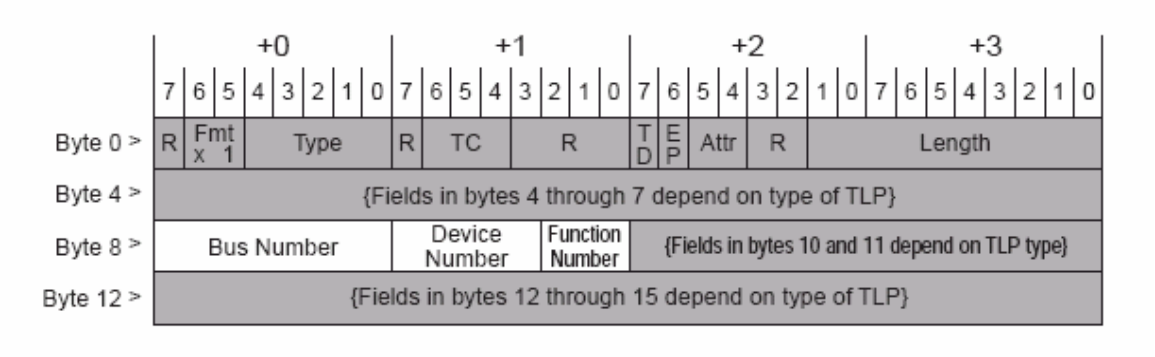


Fig 4-7 ID Routing with 4 DW Header

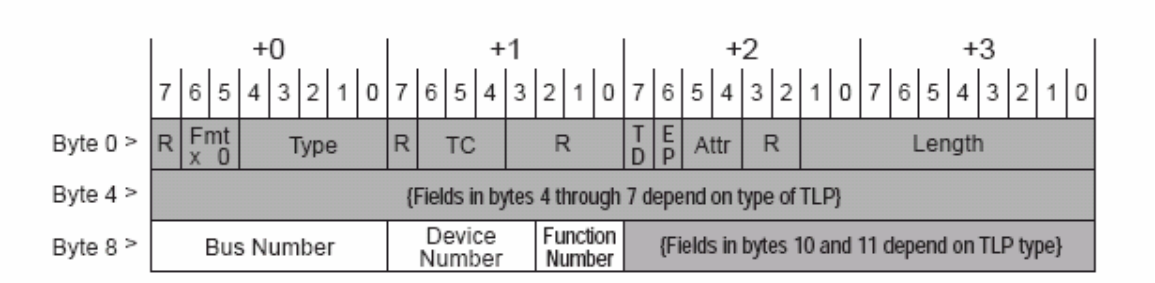


Fig 4-8 ID Routing with 3 DW Header

4.3.5 First/Last DW Byte Enables Rules

Byte Enables are included with Memory, I/O, and Configuration Requests. This section defines the corresponding rules. Byte Enables, when present in the Request header, are located in byte 7 of the header .

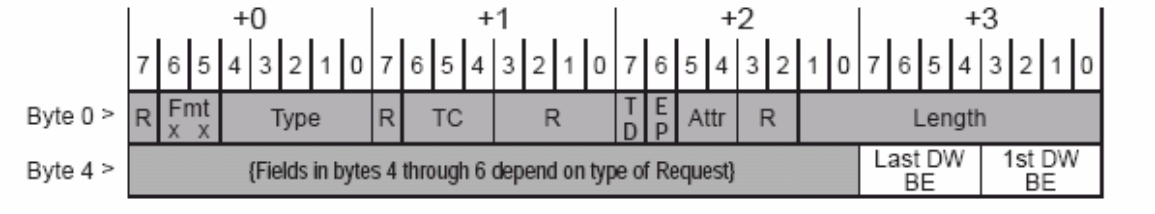


Fig 4-9 Byte Enables

The 1st DW BE[3:0] field contains Byte Enables for the first (or only) DW referenced by a Request.

- If the Length field for a Request indicates a length of greater than 1 DW, this field must not equal 0000b.
- The Last DW BE[3:0] field contains Byte Enables for the last DW of a Request.

- If the Length field for a Request indicates a length of 1 DW, this field must equal 0000b.
- If the Length field for a Request indicates a length of greater than 1 DW, this field must not equal 0000b.
- For each bit of the Byte Enables fields:
- A value of 0b indicates that the corresponding byte of data must not be written or, if non prefetchable, must not be read at the Completer.
- A value of 1b indicates that the corresponding byte of data must be written or read at the Completer.
- Non-contiguous Byte Enables (enabled bytes separated by non-enabled bytes) are permitted in the 1st DW BE field for all Requests with length of 1 DW.
- Non-contiguous Byte Enable examples: 1010b, 0101b, 1001b, 1011b, 1101b
- Non-contiguous Byte Enables are permitted in both Byte Enables fields for QW aligned

Memory Requests with length of 2 DW (1 QW).

- All non-QW aligned Memory Requests with length of 2 DW (1 QW) and Memory Requests with length of 3 DW or more must enable only bytes that are contiguous with the data between the first and last DW of the Request.
- Contiguous Byte Enables examples:
  - 1st DW BE: 1100, Last DW BE: 0011
  - 1st DW BE: 1000, Last DW BE: 0111

Table 4-7 shows the correspondence between the bits of the Byte Enables fields, their location in the Request header, and the corresponding bytes of the referenced data.

4.3.6. Transaction Descriptor

Overview

The Transaction Descriptor is a mechanism for carrying Transaction information between the Requester and the Completer. Transaction Descriptors are composed of three fields:

- Transaction ID – identifies outstanding Transactions 15
- Attributes field – specifies characteristics of the Transaction
- Traffic Class (TC) field – associates Transaction with type of required service

Figure 2-10 shows the fields of the Transaction Descriptor. Note that these fields are shown together to highlight their relationship as parts of a single logical entity. The fields are not contiguous in the packet header.

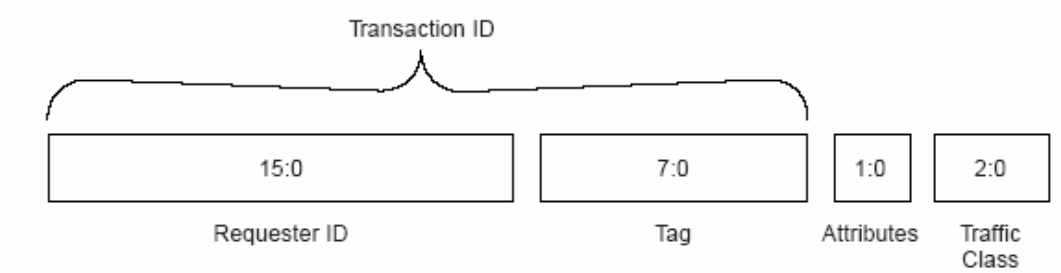


Fig 4-10 Transaction Id

- Tag[7:0] is a 8-bit field generated by each Requestor, and it must be unique for all outstanding Requests that require a Completion for that Requester.
- By default, the maximum number of outstanding Requests per device/function shall be limited to 32, and only the lower 5 bits of the Tag field are used with the remaining upper 3 bits required to be all 0's.
- If the Extended Tag Field Enable bit is set, the maximum is increased to 256, and the entire Tag field is used Receiver/Completer behavior is undefined if multiple Requests are issued non-unique tag values.
- If Phantom Function Numbers are used to extend the number of outstanding requests, the combination of the Phantom Function Number and the Tag field must be unique for all outstanding Requests that require a Completion for that Requester.

- For Requests that do not require a Completion (Posted Requests), the value in the Tag[7:0] field is undefined and may contain any value. for exceptions to this rule for certain Vendor Defined Messages.
- For Posted Requests, the value in the Tag[7:0] field must not affect Receiver processing of the Request .
- Requester ID and Tag combined form a global identifier, i.e., Transaction ID for each Transaction within a Hierarchy.
  - Transaction ID is included with all Requests and Completions.
  - The Requester ID is a 16-bit value that is unique for every PCI Express function within a Hierarchy.

4.3.6.1 Transaction Descriptor – Attributes Field

The Attributes field is used to provide additional information that allows modification of the default handling of Transactions. These modifications apply to different aspects of handling the Transactions within the system, such as:

- Ordering
- Hardware coherency management (snoop)

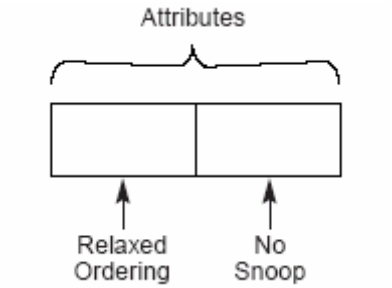


Fig 4-11 Attributes Field of Transaction Descriptor

4.3.6.2 Transaction Descriptor – Traffic Class Field

The Traffic Class (TC) is a 3-bit field that allows differentiation of transactions into eight traffic classes. Together with the PCI Express Virtual Channel support, the TC mechanism is a fundamental element for enabling differentiated traffic servicing. Every PCI Express Transaction Layer Packet uses TC information as an invariant label that is carried end to end within the PCI Express fabric.

As the packet traverses across the fabric, this information is used at every Link and within each Switch element to make decisions with regards to proper servicing of the traffic. A key aspect of servicing is the routing of the packets based on their TC labels through corresponding Virtual Channels. Section 2.5 covers the details of the VC mechanism.

4.3.7 Memory, I/O, and Configuration Request Rules

The following rule applies to all Memory, I/O and Configuration Requests. Additional rules specific to each type of Request follow.

- All Memory, I/O and Configuration Requests include the following fields in addition to the common header fields:
  - Requester ID[15:0] and Tag[7:0], forming the Transaction ID
  - Last DW BE[3:0] and 1st DW BE[3:0]

For **Memory Requests,** the following rules apply:

- Memory Requests route by address, using either 64-bit or 32-bit Addressing
- For Memory Read Requests, Length must not exceed the value specified by Max_Read_Request_Size

Requests must not specify an Address/Length combination which causes a Memory Space access to cross a 4-KB boundary.

- Receivers may optionally check for violations of this rule. If a Receiver implementing this check determines that a TLP violates this rule, the TLP is a Malformed TLP.
- If checked, this is a reported error associated with the Receiving Port

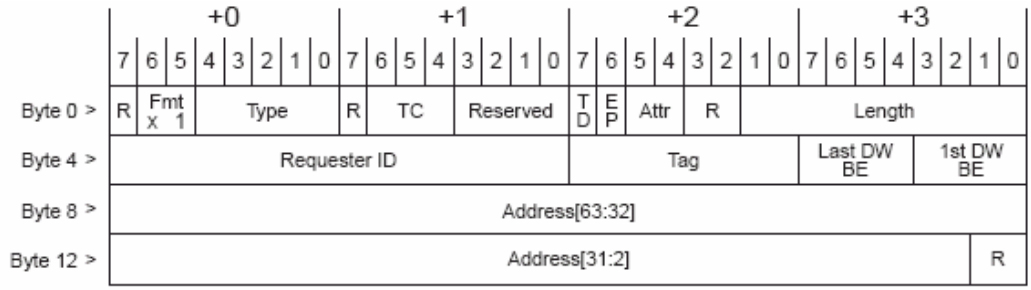


Fig 4-12 Request Header Format for 64-bit Addressing of Memory

For **I/O Requests,** the following rules apply:

- I/O Requests route by address, using 32-bit Addressing I/O Requests have the following restrictions:
    - TC[2:0] must be 000b
    - Attr[1:0] must be 00b
- Length[9:0] must be 00 0000 0001b
- Last DW BE[3:0] must be 0000b
- Receivers may optionally check for violations of these rules. If a Receiver implementing these checks determines that a TLP violates these rules, the TLP is a Malformed TLP

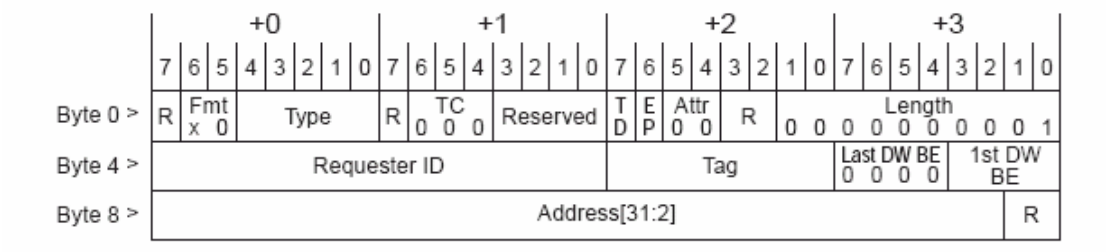


Fig 4-13 Request Header Format for I/O Transactions

For **Configuration Requests**, the following rules apply:

- Configuration Requests route by ID, and use a 3 DW header
- In addition to the header fields included in all Memory, I/O, and Configuration Requests and the ID routing fields, Configuration Requests contain the following additional field
    - Register Number[5:0]
    - Extended Register Number[3:0]
- Configuration Requests have the following restrictions:
    - TC[2:0] must be 000b
    - Attr[1:0] must be 00b
    - Length[9:0] must be 00 0000 0001b
    - Last DW BE[3:0] must be 0000b

Receivers may optionally check for violations of these rules. If a Receiver implementing these checks determines that a TLP violates these rules, the TLP is a Malformed TLP.

4.3.8 Power Management Messages

These Messages are used to support PCI Express power management, which is described in detail in The following rules define the Power Management Messages:

- Table 3-7 defines the Power Management Messages.
- Power Management Messages do not include a data payload (TLP Type is Msg).
- The Length field is reserved.
- Power Management Messages must use the default Traffic Class designator (TC0). Receivers must check for violations of this rule. If a Receiver determines that a TLP violates this rule, it must handle the TLP as a Malformed TLP.

Ta

Name	Code[7:0]	Routing r[2:0]	Support				Req ID	Description/Comments
			RC	Ep	Sw	Br		
ERR_COR	0011 0000	000	r	t	tr	t	BD BDF	This Message is issued when the component or device detects a correctable error on the PCI Express interface.
ERR_NONFATAL	0011 0001	000	r	t	tr	t	BD BDF	This Message is issued when the component or device detects a Non-fatal, uncorrectable error on the PCI Express interface.
ERR_FATAL	0011 0011	000	r	t	tr	t	BD BDF	This Message is issued when the component or device detects a Fatal, uncorrectable error on the PCI Express interface.

Table 4-7 Power management Messages

The initiator of the Message is identified with the Requester ID of the Message header. The Root Complex translates these error Messages into platform level events. Refer to Section 6.2 for details on uses for these Messages.

4.3.8.1 Error Signaling Messages

Error Signaling Messages are used to signal errors that occur on specific transactions and errors that are not necessarily associated with a particular transaction. These Messages are initiated by the agent that detected the error.

- Table 3-18 defines the Error Signaling Messages.
- Error Signaling Messages do not include a data payload (TLP Type is Msg).
- The Length field is reserved.
- Error Signaling Messages must use the default Traffic Class designator (TC0) Receivers must check for violations of this rule. If a Receiver determines that a TLP violates this rule, it must handle the TLP as a Malformed TLP.

Name	Code[7:0]	Routing r[2:0]	Support				Req ID	Description/Comments
			RC	Ep	Sw	Br		
ERR_COR	0011 0000	000	r	t	tr	t	BD BDF	This Message is issued when the component or device detects a correctable error on the PCI Express interface.
ERR_NONFATAL	0011 0001	000	r	t	tr	t	BD BDF	This Message is issued when the component or device detects a Non-fatal, uncorrectable error on the PCI Express interface.
ERR_FATAL	0011 0011	000	r	t	tr	t	BD BDF	This Message is issued when the component or device detects a Fatal, uncorrectable error on the PCI Express interface.

Table 4-8 Error Signaling Messages

4.3.9. Completion Rules

All Read Requests and Non-Posted Write Requests require Completion. Completions include a Completion header that, for some types of Completions, will be followed by some number of DW of data. The rules for each of the fields of the Completion header are defined in the following sections.

- Completions route by ID, and use a 3 DW header

Note that the routing ID fields correspond directly to the Requester ID supplied with the Corresponding Request. Thus for Completions these fields will be referred to collectively as the Requester ID instead of the distinct fields used generically for ID routing.

- In addition to the header fields included in all TLPs and the ID routing fields, Completions contain the following additional fields Completer ID[15:0] – Identifies the Completer – described in detail below.
- Completion Status[2:0] – Indicates the status for a Completion .
- Rules for determining the value in the Completion Status[2:0] field are discussed.
- BCM – Byte Count Modified – this bit must not set by PCI Express Completers, and may only be set by PCI-X completers
- Byte Count[11:0] – The remaining byte count for Request
- The Byte Count value is specified as a binary number, with 0000 0000 0001b indicating 1 byte, 1111 1111 1111b indicating 4095 bytes, and 0000 0000 0000b indicating 4096 bytes
- For Memory Read Completions, Byte Count[11:0] is set according to the rules .

For all other types of Completions, the Byte Count field must be 4.

- Tag[7:0] in combination with the Requester ID field, corresponds to the Transaction ID
- Lower Address[6:0] – lower byte address for starting byte of Completion
- For Memory Read Completions, the value in this field is the byte address for the first enabled byte of data returned with the Completion

This field is set to all 0's for all types of Completions other than Memory Read Completions

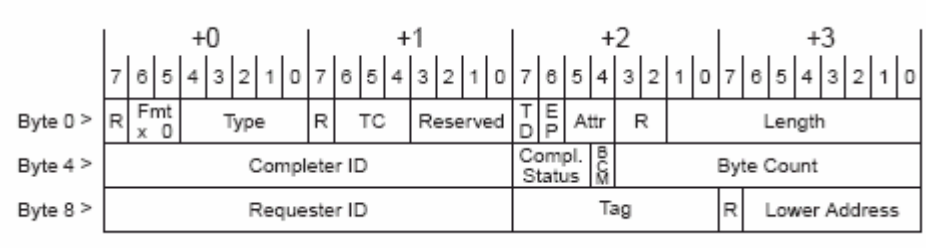


Fig 4-14 Completion Header Format

Completion Status[2:0] Field Value	Completion Status
000b	Successful Completion (SC)
001b	Unsupported Request (UR)
010b	Configuration Request Retry Status (CRS)
100b	Completer Abort (CA)
all others	Reserved

Table 4-9 completion status

The Completer ID[15:0] is a 16-bit value that is unique for every PCI Express function within a Hierarchy

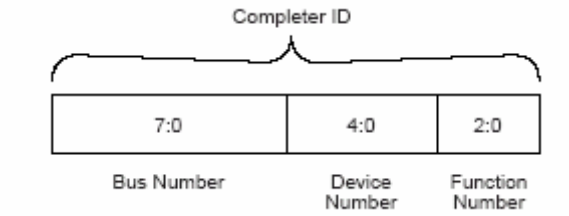


Fig 4-15 Completer ID

Functions must capture the Bus and Device Numbers supplied with all Type 0 Configuration Write Requests completed by the function, and supply these numbers in the Bus and Device Number fields of the Completer ID for all Completions generated by the device/function.

- If a function must generate a Completion prior to the initial device Configuration Write Request, 0's must be entered into the Bus Number and Device Number fields

- Note that Bus Number and Device Number may be changed at run time, and so it is necessary to re-capture this information with each and every Configuration Write Request.

  Exception: The assignment of bus numbers to the logical devices within a Root Complex may be done in an implementation specific way.

In some cases, a Completion with the UR status may be generated by a multi-function device without associating the Completion with a specific function within the device – in this case, the Function Number field is Reserved.

Example: A multi-function device receives a Read Request which does not target any resource associated with any of the functions of the device – the device generates a 5 Completion with UR status and sets a value of all 0's in the Function Number field of the Completer ID

- Completion headers must supply the same values for the Requester ID, Tag, Attribute and Traffic Class as were supplied in the header of the corresponding Request.
- The Completion ID field is not meaningful prior to the software initialization and configuration of the completing device (using at least one Configuration Write Request), and the Requestor must ignore the value returned in the Completer ID field.
- A Completion including data must specify the actual amount of data returned in that Completion, and must include the amount of data specified.
- It is a TLP formation error to include more or less data than specified in the Length field, and the resulting TLP is a malformed TLP.

Note: This is simply a specific case of the general rule requiring TLP data payload length match the value in the Length field

3.4 Handling of Received TLPs

This section describes how all Received TLPs are handled when they are delivered to the Receive Transaction Layer from the Receive Data Link Layer, after the Data Link Layer has validated the integrity of the received TLP. The rules are diagramed in the flowchart shown in Figure 3-16.

- Values in Reserved fields must be ignored by the Receiver.
- All Received TLPs which use undefined Type field values are Malformed TLPs.

- This is a reported error associated with the Receiving Port
- All Received Malformed TLPs must be discarded.
- Received Malformed TLPs that are ambiguous with respect to which buffer to release or are mapped to an uninitialized virtual channel must be discarded without updating Receiver Flow Control information.
- All other Received Malformed TLPs must be discarded, optionally not updating Receiver Flow Control information.
- Otherwise, update Receiver Flow Control tracking information
- If the value in the Type field indicates the TLP is a Request, handle according to Request Handling Rules, otherwise, the TLP is a Completion – handle according to Completion handling Rules (following sections)

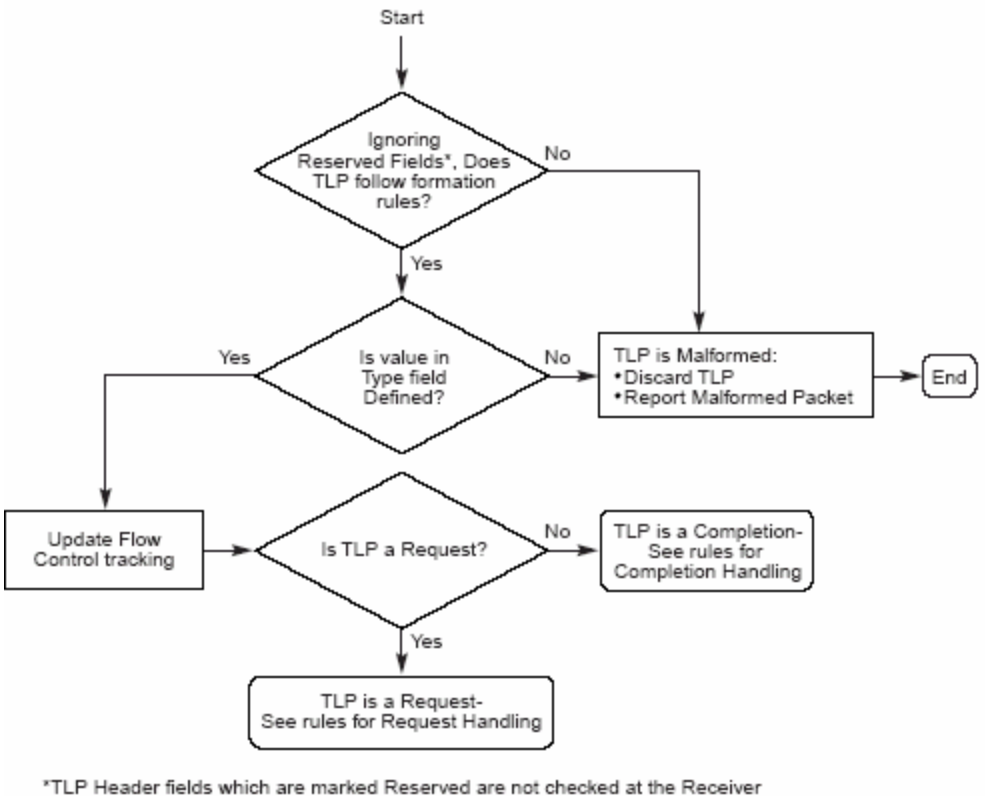


Fig 4-16 Flowchart for Handling of Received TLPs

Switches must process both TLPs which address resources within the Switch as well as TLPs which address resources residing outside the Switch. Switches handle all TLPs

which address internal resources of the Switch according to the rules above. TLPs which pass through the Switch, or which address the Switch as well as passing through it, are handled according to the following rules (see Figure 3-17)

- If the value in the Type field indicates the TLP is not a Msg or MsgD Request, the TLP must be routed according to the routing mechanism used .
- Switches route Completions using the information in the Requester ID field of the Completion.

If the value in the Type field indicates the TLP is a Msg or MsgD Request, route the Request according to the routing mechanism indicated in the r[2:0] sub-field of the Type field

- If the value in r[2:0] indicates the Msg/MsgD is routed to the Root Complex (000b), the Switch must route the Msg/MsgD to the Upstream Port of the Switch
- It is an error to receive if a Msg/MsgD Request specifying 000b routing at the Upstream Port of a Switch. Switches may check for violations of this rule – TLPs in violation are Malformed TLPs. If checked, this is a reported error associated with the Receiving Port
- If the value in r[2:0] indicates the Msg/MsgD is routed by address (001b), the Switch must route the Msg/MsgD in the same way it would route a Memory Request by address
- If the value in r[2:0] indicates the Msg/MsgD is routed by ID (010b), the Switch must route the Msg/MsgD in the same way it would route a Completion by ID
- If the value in r[2:0] indicates the Msg/MsgD is a broadcast from the Root Complex (011b), the Switch must route the Msg/MsgD to all Downstream Ports of the Switch .It is an error to receive a Msg/MsgD Request specifying 011b routing at the Downstream Port of a Switch. Switches may check for violations of this rule – TLPs in violation are Malformed TLPs. If checked, this is a reported error associated with the Receiving Port
- If the value in r[2:0] indicates the Msg/MsgD terminates at the Receiver (100b or a reserved value), or if the Message Code field value is defined and corresponds to a Message which must be comprehended by the Switch, the Switch must process the Message according to the Message processing rules

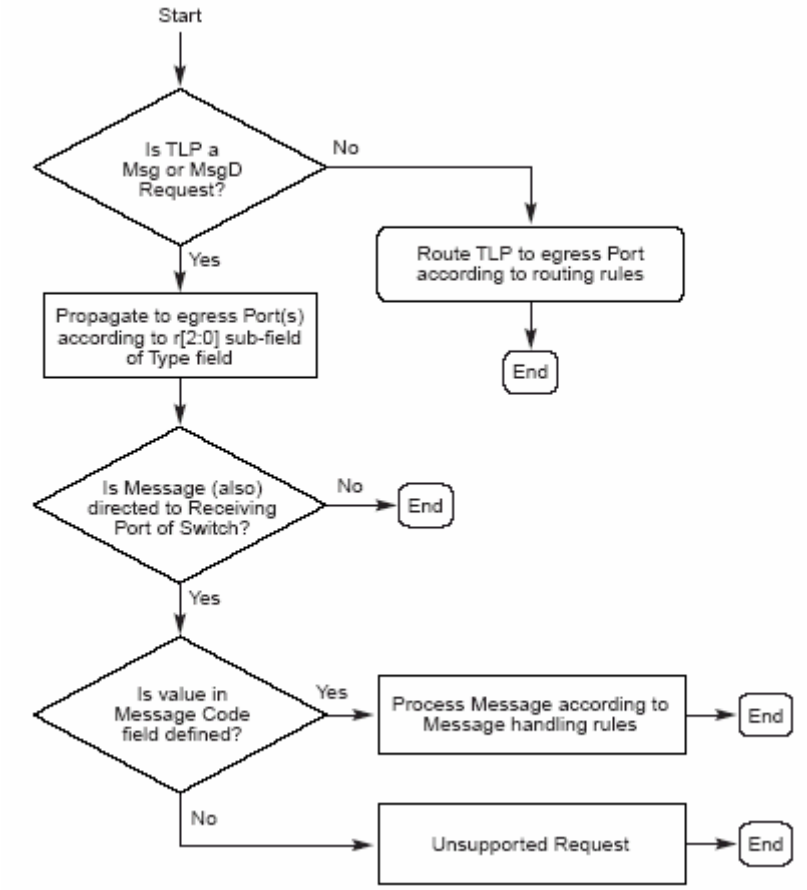


Fig 4-17 Flow Chart For Switch Handling Of Tlps

4.4.1 Data Return for Read Requests

Individual Completions for Memory Read Requests may provide less than the full amount of data Requested so long as all Completions for a given Request when combined return exactly the amount of data Requested in the Read Request.

- Completions for different Requests cannot be combined.
- I/O and Configuration Reads must be completed with exactly one Completion.

The Completion Status for a Completion corresponds only to the status associated with the data returned with that Completion

- A Completion with status other than Successful Completion terminates the Completionsfor a single Read Request . In this case, the value in the Length field is undefined, and must be ignored by the Receiver.
- Completions must not include more data than permitted by the Max_Payload_Size parameter. Receivers must check for violations of this rule.

Note: This is simply a specific case of the rules which apply to all TLPs with data payloads

Memory Read Requests may be completed with one, or in some cases, multiple Completions

The Read Completion Boundary (RCB) parameter determines the naturally aligned address boundaries on which a Read Request may be serviced with multiple Completions

- For a Root Complex, RCB is 64 bytes or 128 bytes
- This value is reported through a configuration register (see Section 7.8)

Note: Bridges and Endpoints may implement a corresponding command bit which may be set by system software to indicate the RCB value for the Root Complex, allowing the Bridge/Endpoint to optimize its behavior when the Root Complex's RCB is 128 bytes.

- For all other system elements, RCB is 128 bytes
- Completions for Requests which do not cross the naturally aligned address boundaries at integer multiples of RCB bytes must include all data specified in the Request

Requests which do cross the address boundaries at integer multiples of RCB bytes may be completed using more than one Completion, but the data must not be fragmented except along the following address boundaries:

- The first Completion must start with the address specified in the Request, and must end at one of the following:
    - The address specified in the Request plus the length specified by the Request (i.e., the entire Request)
    - An address boundary between the start and end of the Request at an integer multiple of RCB bytes
    - The final Completion must end with the address specified in the Request plus the length specified by the Request
    - All Completions between, but not including, the first and final Completions must be an integer multiple of RCB bytes in length

➢ Receivers may optionally check for violations of RCB. If a Receiver implementing this check determines that a Completion violates this rule, it must handle the Completion as a Malformed TLP

- This is a reported error associated with the Receiving Port
- Multiple Memory Read Completions for a single Read Request must return data in increasing address order.
- For each Memory Read Completion, the Byte Count field must indicate the remaining number of bytes required to complete the Request including the number of bytes returned with the Completion, except when the BCM field is 1 .

The total number of bytes required to complete a Memory Read Request is calculated as shown in Table 3-9 .

- If a Memory Read Request is completed using multiple Completions, the Byte Count value for each successive Completion is the value indicated by the preceding Completion minus the number of bytes returned with the preceding Completion.

For all Memory Read Completions, the Lower Address field must indicate the lower bits of the byte address for the first enabled byte of data returned with the Completion

- For the first (or only) Completion, the Completer can generate this field from the least significant five bits of the address of the Request concatenated with two bits of byte-level address formed as shown in Table 3-9
- For any subsequent Completions, the Lower Address field will always be zero except for Completions generated by a Root Complex with an RCB value of 64 bytes. In this case the least significant 6 bits of the Lower Address field will always be zero and the most significant bit of the Lower Address field will toggle according to the alignment of the 64-byte data payload.

Table 4-9 Calculating Byte Count from Length and Byte Enables

1st DW BE[3:0]	Last DW BE[3:0]	Total Byte Count
1xx1	0000[10]	4
01x1	0000	3
1x10	0000	3
0011	0000	2
0110	0000	2
1100	0000	2
0001	0000	1
0010	0000	1
0100	0000	1
1000	0000	1
0000	0000	1
xxx1	1xxx	Length[11] * 4
xxx1	01xx	(Length * 4) - 1
xxx1	001x	(Length * 4) - 2
xxx1	0001	(Length * 4) - 3
xx10	1xxx	(Length * 4) - 1
xx10	01xx	(Length * 4) - 2
xx10	001x	(Length * 4) - 3
xx10	0001	(Length * 4) - 4
x100	1xxx	(Length * 4) - 2
x100	01xx	(Length * 4) - 3
x100	001x	(Length * 4) - 4
x100	0001	(Length * 4) - 5
1000	1xxx	(Length * 4) - 3
1000	01xx	(Length * 4) - 4
1000	001x	(Length * 4) - 5
1000	0001	(Length * 4) - 6

Table 4-10 Calculating Lower Address from 1st DW BE

1st DW BE[3:0]	Lower Address[1:0]
0000	00b
xxx1	00b
xx10	01b
x100	10b
1000	11b

When a Read Completion is generated with a Completion Status other than Successful Completion:

- No data is included with the Completion
- The Cpl (or CplLk) encoding is used instead of CplD (or CplDLk)
- This Completion is the final Completion for the Request.
- The Completer must not transmit additional Completions for this Request.
- Example: Completer split the Request into four parts for servicing; the second completion had a Completer Abort Completion Status; the Completer terminated Servicing for the Request, and did not transmit the remaining two Completions.
- The Byte Count field must indicate the remaining number of bytes that would be required to complete the Request (as if the Completion Status were Successful Completion)
- The Lower Address field must indicate the lower bits of the byte address for the first enabled byte of data that would have been returned with the Completion if the Completion Status were Successful Completion

4.5 Transaction Ordering

4.5.1. Transaction Ordering Rules

Table 4-11 defines the ordering requirements for PCI Express Transactions. The rules defined in this table apply uniformly to all types of Transactions on PCI Express including Memory, I/O, Configuration, and Messages. The ordering rules defined in this table apply within a single Traffic Class (TC). There is no ordering requirement among transactions with different TC labels.

Note: that this also implies that there is no ordering required between traffic that flows through different Virtual Channels since transactions with the same TC label are not allowed to be mapped to multiple VCs on any PCI Express Link.

For Table 4-11, the columns represent a first issued transaction and the rows represent a subsequently issued transaction. The table entry indicates the ordering relationship between the two transactions.

The table entries are defined as follows:

- Yes–the second transaction (row) must be allowed to pass the first (column) to avoid deadlock. (When blocking occurs, the second transaction is required to pass the first transaction. Fairness must be comprehended to prevent starvation.)
- Y/N–there are no requirements. The second transaction may optionally pass the first transaction or be blocked by it.
- No–the second transaction must not be allowed to pass the first transaction. This is required to support the Producer-Consumer strong ordering model.

Table 4-11 Ordering Rules Summary Table

Row Pass Column?		Posted Request	Non-Posted Request		Completion	
		Memory Write or Message Request (Col 2)	**Read Request** (Col 3)	**I/O or Configuration Write Request** (Col 4)	**Read Completion** (Col 5)	**I/O or Configuration Write Completion** (Col 6)
Posted Request	**Memory Write or Message Request** (Row A)	a) No b) Y/N	Yes	Yes	a) Y/N b) Yes	a) Y/N b) Yes
Non-Posted Request	**Read Request** (Row B)	No	Y/N	Y/N	Y/N	Y/N
	I/O or Configuration Write Request (Row C)	No	Y/N	Y/N	Y/N	Y/N
Completion	**Read Completion** (Row D)	a) No b)Y/N	Yes	Yes	a) Y/N b) No	Y/N
	I/O or Configuration Write Completion (Row E)	Y/N	Yes	Yes	Y/N	Y/N

4.6 Ordering and Receive Buffer Flow Control

Flow Control (FC) is used to prevent overflow of Receiver buffers and to enable compliance with the ordering rules defined in Section 2.4. Note that the Flow Control mechanism is used by the Requester to track the queue/buffer space available in the Agent across the Link as shown before. That is, Flow Control is point-to-point (across a Link) and not end-to-end. Flow Control does not imply that a Request has reached its ultimate Completer.

Fig 4-19 Relationship Between Requester and Ultimate Completer

- Flow Control is orthogonal to the data integrity mechanisms used to implement reliable information exchange between Transmitter and Receiver. Flow Control can treat the flow of TLP information from Transmitter to Receiver as perfect, since the data integrity mechanisms ensure that corrupted and lost TLPs are corrected through retransmission
- Each Virtual Channel maintains an independent Flow Control credit pool. The FC information is conveyed between two sides of the Link using DLLPs. The VC ID field of the DLLP is used to carry the Virtual Channel Identification that is required for proper flow-control credit accounting.
- Flow Control mechanisms used internally within a multi-function device are outside the scope of this specification. Flow Control is handled by the Transaction Layer in cooperation with the Data Link Layer.
- The Transaction Layer performs Flow Control accounting functions for Received TLPs and "gates" TLP Transmissions based on available credits for transmission.

In this and other sections of this specification, rules are described using conceptual "registers" that a PCI Express device could use in order to implement a PCI Express compliant design. This description does not imply or require a particular implementation and is used only to clarify the requirements.

- Flow Control information is transferred using Flow Control Packets (FCPs), which are a type of DLLP
- The unit of Flow Control credit is 4 DW for data
- For headers, the unit of Flow Control credit is one maximum-size header plus TLP digest
- Each Virtual Channel has independent Flow Control
- Flow Control distinguishes three types of TLPs (note relationship to ordering rules • Posted Requests (P) – Messages and Memory Writes

- Non-Posted Requests (NP) – All Reads, I/O, and Configuration Writes
- Completions (CPL) – Associated with corresponding NP Requests

In addition, Flow Control distinguishes the following types of TLP information within each of the three types:

- Headers (H)
- Data (D)

Table 4-10 Flow Control Credit Types

Credit Type	Applies to This Type of TLP Information
PH	Posted Request headers
PD	Posted Request Data payload
NPH	Non-Posted Request headers
NPD	Non-Posted Request Data payload
CPLH	Completion headers
CPLD	Completion Data payload

4.7 Data Integrity

The basic data reliability mechanism in PCI Express is contained within the Data Link Layer, which uses a 32-bit CRC (LCRC) code to detect errors in TLPs on a Link-by-Link basis, and applies a Link-by-Link retransmit mechanism for error recovery. A TLP is a unit of data and transaction control that is created by a data-source at the "edge" of the PCI Express domain (such as an Endpoint or Root Complex), potentially routed through intermediate components (i.e., Switches) and consumed by the ultimate PCI Express recipient. As a TLP passes through a Switch, the Switch may need to change some control fields without modifying other fields that should not change as the packet traverses the path. Therefore, the LCRC is regenerated by Switches. Data corruption may occur internally to the Switch, and the regeneration of a good LCRC for corrupted data masks the existence of errors. To ensure end-to-end data integrity detection in systems that require high data reliability, a Transaction Layer end-to-end 32-bit CRC (ECRC) can be placed in the TLP Digest field at the end of a TLP. The ECRC covers all fields that do not change as the TLP traverses the path (invariant fields). The ECRC is generated by the Transaction Layer in the source component, and checked in the destination component. A Switch that supports ECRC checking checks ECRC on TLPs that are destined to a destination within the Switch

itself. On all other TLPs a Switch must preserve the ECRC (forward it untouched) as an integral part of the TLP. In some cases, the data in a TLP payload is known to be corrupt at the time the TLP is generated, or may become corrupted while passing through an intermediate component, such as a Switch. In these cases, error forwarding, also known as data poisoning, can be used to indicate the corruption to the device consuming the data.

4.7.1. ECRC Rules

The capability to generate and check ECRC is reported to software, and the ability to do so is enabled by software

- If a device is enabled to generate ECRC, it must calculate and apply ECRC for all TLPs originated by the device
- Switches must pass TLPs with ECRC unchanged from the Ingress Port to the Egress Port
- If a device reports the capability to check ECRC, it must support Advanced Error Reporting
- If a device is enabled to check ECRC, it must do so for all TLPs received by the device including ECRC
- Note that it is still possible for the device to receive TLPs without ECRC, and these areprocessed normally – this is not an error
- Note that a Switch may perform ECRC checking on TLPs passing through the Switch. ECRC
- Errors detected by the Switch are reported but do not alter the TLPs passage through the Switch.
- A 32-bit ECRC is calculated for the entire TLP (header and data payload) using the following algorithm and appended to the end of the TLP
- The ECRC value is calculated using the following algorithm
- The polynomial used has coefficients expressed as 04C1 1DB7h
- The seed value (initial value for ECRC storage registers) is FFFF FFFFh
- All invariant fields of the TLP header and the entire data payload (if present) are included in the ECRC calculation, all bits in variant fields must be set to 1 for ECRC calculations.

- Bit 0 of the Type field is variant The EP field is variant all other fields are invariant
- ECRC calculation starts with bit 0 of byte 0 and proceeds from bit 0 to bit 7 of each byte of the TLP

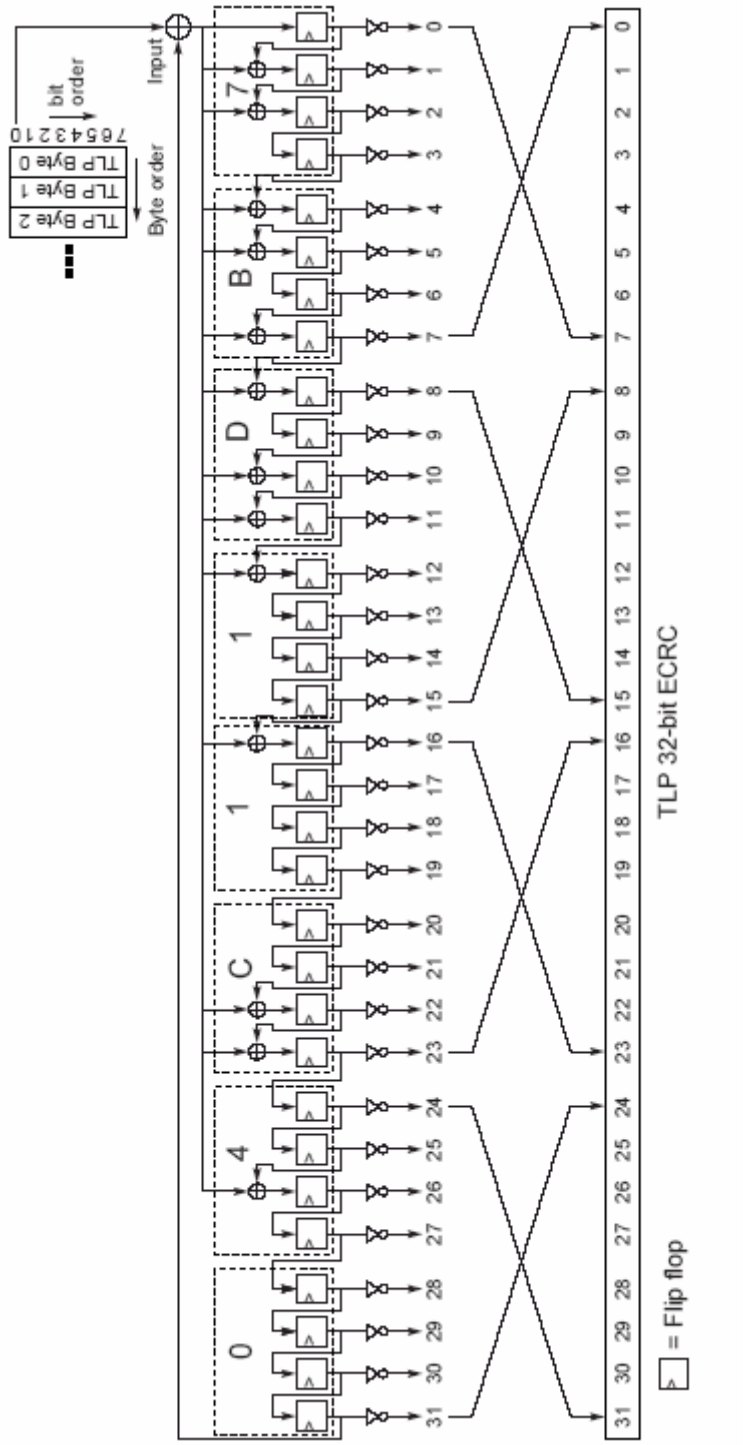


Fig 4-20 Calculation Of 32-Bit Ecrc For Tlp End To End Data Integrity Protection

4.7.2 Completion Timeout Mechanism

In any split transaction protocol, there is a risk associated with the failure of a Requester to receive an expected Completion. To allow Requesters to attempt recovery from this situation in a standard manner, the Completion Timeout mechanism is defined. This mechanism is intended to be activated only when there is no reasonable expectation that the Completion will be returned, and should never occur under normal operating conditions. Note that the values specified here do not reflect expected service latencies, and must not be used to estimate typical response times.

PCI Express devices that issue Requests requiring Completions must implement the Completion Timeout mechanism. An exception is made for Configuration Requests (see below). The Completion Timeout mechanism is activated for each Request that requires one or more Completions when the Request is transmitted. Since PCI Express Switches do not autonomously initiate Requests that need Completions, the requirement for Completion Timeout support is limited only to Root Complexes, PCI Express-PCI Bridges, and Endpoint devices.

This specification defines the following range for the minimum/maximum acceptable timer values for the Completion Timeout mechanism:

- The Completion Timeout timer must not expire (i.e., cause a timeout event) in less than 50 μs. It is strongly recommend that unless an application requires this level of timer granularity the minimum time should not expire in less than 10 ms.
- The Completion Timeout timer must expire if a Request is not completed in 50 ms.

Chapter 5
IMPLEMENTATION

5.1 PCIe Block Diagram

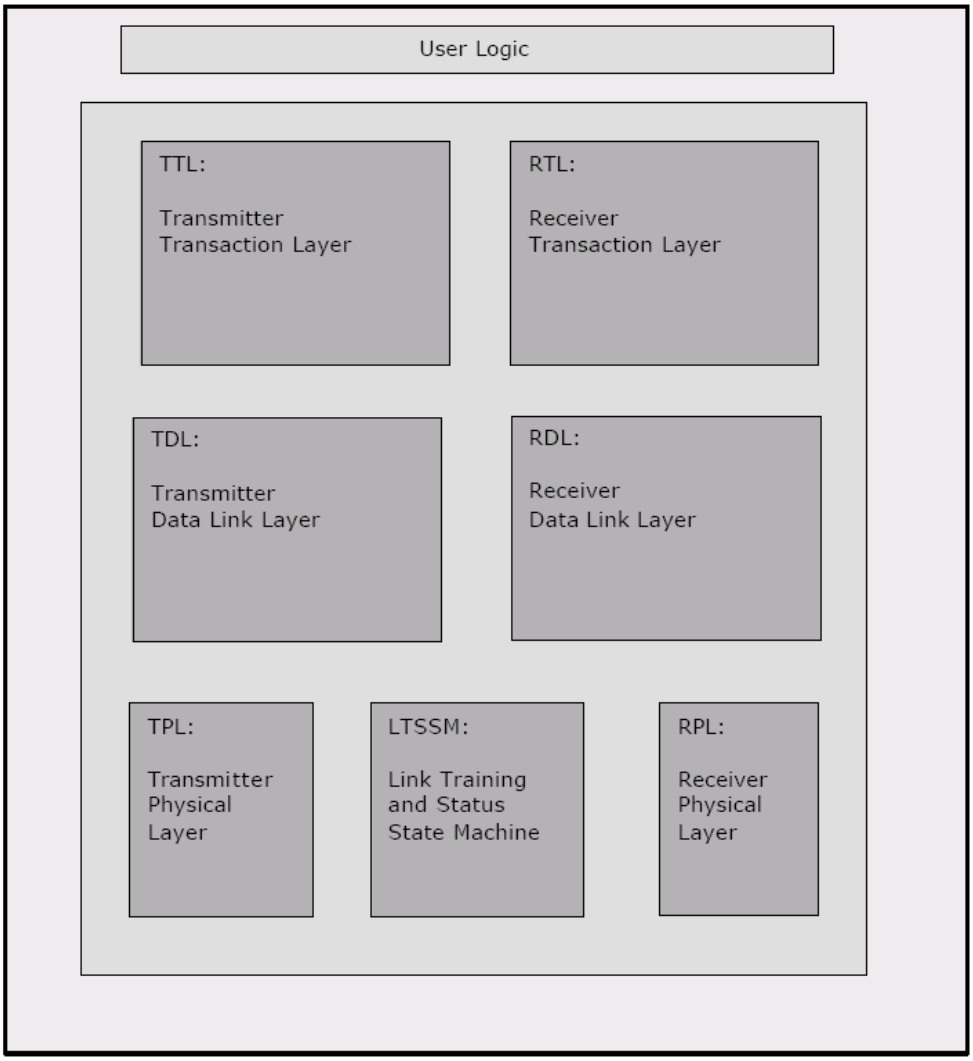


Fig 5-1 PCIe block Diagram

In this thesis an approach to design the Transaction layer of PCI Express is given and it is implemented using Verilog- HDL and it is verified by designing a configurable Application Layer named as Application Bus Function Model (ABFM) .

5.2 APPLICATION BUS FUNCTION MODEL(ABFM)

The user interface for PCI Express is named as ABFM . It describes interface signals for sending transactions from User Logic to core as well as receiving from core to User Logic. This module interacts with the Transaction Layer . As you see from the Block diagram above It shows the Transaction Layer divided into two parts namely

- Transmit Transaction Layer (TTL)
- Receive Transaction Layer (RTL)

In the same way the Application Layer is also divided into two Parts namely ,

- Application Bus Function - Master
- ABFM – Slave.

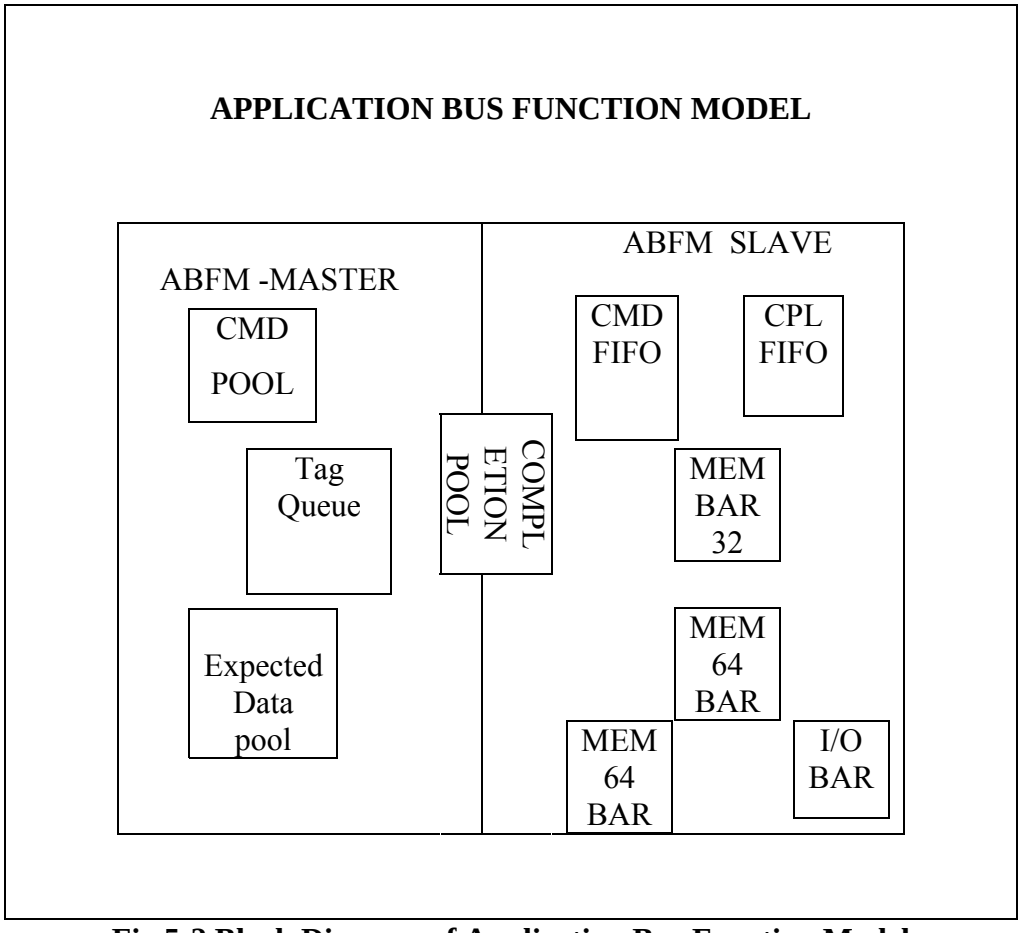


Fig 5-2 Block Diagram of Application Bus Function Model

5.2.1 ABFM –MASTER

This block Interacts with TTL ,ABFM-MASTER is responsible to initiate transactions on to TTL .ABFM- Master transmits the Header and Data on to TTL .This Is also responsible to transmit the Completion packets prepared by ABFM-Slave .

ABFM master maintains a queue named TAG_QUEUE which is used to store the assigned TAG to the non-posted Packets transmitted by ABFM-Master .This also maintains a QUEUE where the expected data to be returned by the device standing at the far end Is stored for data –checking purpose when a READ transaction followed by a WRITE transaction is Transmitted by ABFM-MASTER.

Description :

Initially the command to be fired onto the TTL is given to ABFM-MASTER through Test cases which calls the Tasks (a Verilog construct just like a function in C …but task cant written a value while a function a can return a value) written inside the ABFM-Master. This specifies the job to be done by the Master .

A task named **do_cmd** is used to specify the command,

Syntax : do_cmd (parameter_name, Parameter value) ;

Example – do_cmd (`TLP_TYPE,MWR);

This specifies the TLP to be fired is Memory Write

A task named **do_cfg** is used to configure packets transmitted by master.

Syntax : do_cfg (Parameter_name,Parameter_value) ;

Example- do_cfg (`TLP_BYTE_COUNT,20);

This configures the TLP header should have the length as 5DW.

A task named **do_pkt** is used to set the data along with the TLP.

Example : do_pkt(TLP_DATA,INCR);

This specifies the data to be transmitted should be incremental.

Command Pool

This is a data structure implemented which has a configurable storage space, It is basically used here as a FIFO (First in First out).It has a 32-bit registers.All the commands passed through test cases by using do_cmd and do_cfg's are stored inside this pool in the same order in which they are called .later one after the other are used by ABFM-Master to prepare Packets.

Tag queue

This is also a FIFO .According to PCI-Express All the non-posted requests namely MRD and IORD are assigned a unique Tag which is a 7-bit field specified inside the header of all TLP's .To maintain uniqueness the Tag queue is used to store the value of tag whenever assigned.

Expected data queue

This is basically a queue which is used for verification purpose.

Whenever a non-posted type request is fired by ABFM then the data return from the device connected on the other side is written inside the Expected data queue, when data is Returned with the help of the Tag the data is matched and data is compared. If data does not match with the Expected once .then it is intimated as a Error

5.2.2 ABFM-SLAVE

This block interacts with RTL, ABFM-Slave is responsible to receive the TLPs forwarded by RTL towards user, This block implements 8 BAR memories of configured sizes, they can be pointed by the signal named usr_rcv_bar_hit.

Depending on the address specified in the header of the received TLP data on the data bus is written into memory. This block implements a data structure named Pool, which is shared by ABFM-Master. Whenever a packet with out data is received by the ABFM-Slave .Then completion packet is prepared by slave and it is stored inside the completion pool. ABFM Master is responsible to pick out the packets from the completion pool and they are transmitted towards TTL .

Cmd FIFO

This is used to store the commands .This has 32-bit registers , the values inside the register decides the Slave job ,either ACK should be asserted for the TLP or RETRY or DISCARD should be asserted and also specifies when should assert WAIT . All the signals in the interface driven by ABFM-SLAVE are configurable using test cases.

Cpl Fifo

This has 32-bit register's in a queue, Values inside each register specifies the Completion Status in the Completion Packets prepared by ABFM_SLAVE. The values inside the CPL_FIFO registers are specified by do_cfg's written in test cases.

Completion Pool :

This has 32-bit registers aligned .As explained previously This is a data structure. This pool is used to store the completion packets prepared by ABFM_SLAVE in response to the Non-Posted requests received by ABFM_SLAVE.

Note : This Pool is also sharable by ABFM-Master .Master is responsible to delete packets from the Pool.

Base Address Registers (BAR):

These are the Memory's designed inside the Application layer .These are configured by using defines as I/O or MEM bars. When ever Memory Transaction is received then data associated is written into MEM BARs ,if the received is non-Posted memory transaction then the completion packets are made using the MEM BARS. Similarly I/O BARS are accessed for I/O transactions.

5.3 ABFM- MONITOR:

This block is instantiated on the signal interface between the ABFM and TTL and also RTL .This module has two state machines one tracks the ABFM-TTL interface and the other tracks the ABFM-RTL interface. This ***module prints all the transactions*** taking place on the Signal interface and gives the summary at the end of the simulation.

5.4 ABFM-CHECKER:

This block is instantiated on the signal interface between the ABFM and TTL and also RTL. This module also has two state machines one tracks the ABFM-TTL interface and the other tracks the ABFM-RTL interface. This ***module checks the signal interface violations and displays the Errors correspondingly.***

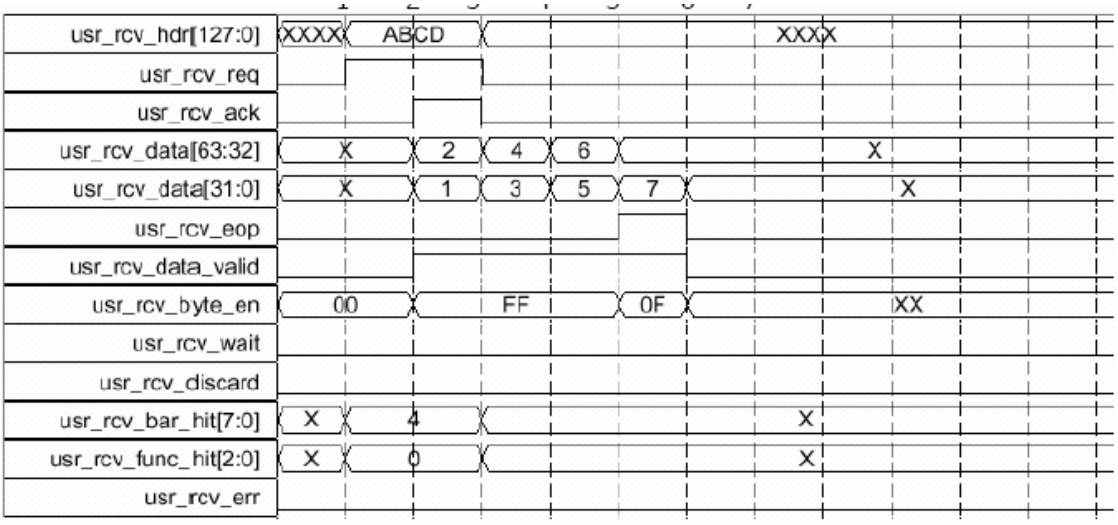


Fig 5-3 Transaction with data payload

Critical Clocks	Event Description
1	• Core places header information of packet on usr_rcv_hdr[127:0], bar decode information on the signal usr_rcv_bar_hit[7:0], function decode information on usr_rcv_func_hit[2:0] and asserts usr_rcv_req to request a transmission
2	• User logic acknowledges the request by asserting usr_rcv_ack • Core places data on data bus usr_rcv_data[63:0] and asserts data valid signal usr_rcv_data_valid
3	• Core deasserts its request usr_rcv_req on receiving acknowledgement from user logic • Core drives the data bus with next two DW's
5	• Core places last data on usr_rcv_data[63:0] • usr_rcv_data[63:32] is don't care in this case • Core also asserts end of packet usr_rcv_eop for one clock cycle to indicate last transfer

Fig 5-4 Transaction with DPl clock description

For better understanding , a brief functional description is given of different blocks described in the PCIe Block Diagram.

5.5 Functionality of PCIe block

5.5.1 Transmission Transaction Layer (TTL)

Primary function of the TTL (Transmitter Transaction Layer) is to receive the TLPs and error signals from User Logic and RTL (Receiver Transaction Layer) which are then forwarded to TDL (Transmitter Data Link Layer). It also generates error message TLPs as requested by all layers. It performs a series of checks for TLPs to be sent e.g TLPs, for which credits are not available or those have value of length greater than Max_pay_load_size are not allowed to pass.

5.5.2 Transmission Data Link Layer (TDL)

This block receives TLPs from TTL and stores them in the Retry Buffer on the fly. It prepends sequence number and appends LCRC to TLP received from TTL or from Retry Buffer during replay. This block also generates four types of DLLPs: Ack/Nak, PM, UFCs and INIT -FCs. 64-bit(x4) data slot of LLTP is transmitted to TPL when ready to receive data from TDL.

5.5.3 Transmission Physical Layer (TPL)

This block deals with the transmission of Transaction layer packets (TLPs), Data Link Layer Packets (DLLPs) and Ordered Sets (OSs) on to the PCIe bus. The LLTP (TLP/DLLP) stream is received from TDL which is distributed on the configured lane(s). The data to be transmitted on each lane is then scrambled and passed on to PHY for transmission on the PCIe bus. The OSs received from LTSSM is passed on directly to PHY.

5.5.4 Link Training and Status State Machine (LTSSM)

This block is the Link Training and Status State Machine (LTSSM) part of the physical layer. It deals with exchange of ordered sets with the other device to form a configured link on which the useful data is transmitted.

5.5.5 Receive Physical Layer (RPL)

This block deals with the receiving of Transaction layer packets (TLPs) and Data Link Layer Packets (DLLPs) from the PHY. The LLTP (TLP/DLLP) stream received from PHY is processed to form valid LLTPs which are presented to RDL for further processing.

5.5.6 Receive Data Link Layer (RDL)

This block receives LLTP (TLP or DLLP) from the RPL in the form of 64-bit(x4) data Slots with the signal indicating whether the received data is part of TLP or DLLP. The integrity and sequence number of the received LLTP is checked and processed.

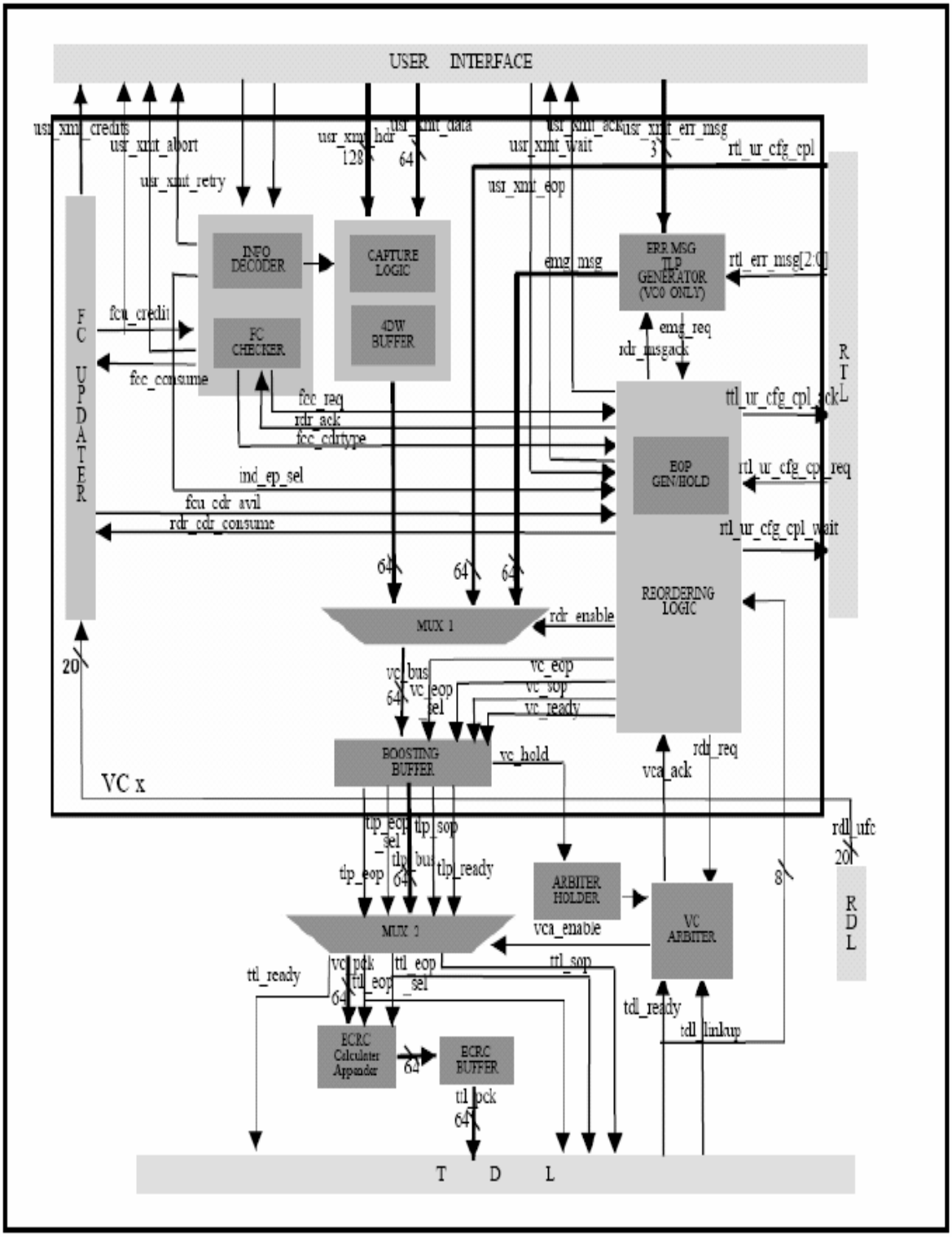
5.5.7 Receive Transaction Layer (RTL)

The primary function of Receive Transaction Layer (RTL) block is to receive TLPs from Receive Data Link Layer (RDL) and pass them on to the User Logic. RTL block performs a series of checks (malformed TLP, UR or Configuration Request) on TLP received from RDL before passing it to user logic. Completions generated for Configuration Request and UR are passed on to the TTL block for transmission. Configuration Requests which require access to user defined configuration area are passed to User Logic. It updates credit information for all types of received TLPs.

RTL block decodes the received TLP for BAR hit and function hit and makes this information available to User Logic along with the corresponding TLP. RTL block also takes care of PCI Express ordering rules for transferring TLP to User Logic.

5.6 Design of Transmission Transaction Layer (TTL)

Fig 5-5 Block Diagram Transmission Transaction Layer



5.6.1 Description of block diagram

5.6.1.1 Info-Decoder

Info-Decoder extracts following information from header of the TLP.

- Header is either 3DW or 4DW wide
- Valid data in the first slot of TLP is either QW or DW
- If the value in length field is greater than the max_pay_load_size then the transaction is aborted by assertion of usr_xmt_abort signal Information extracted from first two points is used by Capture Logic for latching 64- bit slots of TLP in the buffer.

5.6.1.2 FC-Checker

- . Checks whether the credits required to process the transaction is greater than the available credits. The request signal is forward ed to reordering logic otherwise retry signal is asserted.

5.6.1.3 FC-Updater

- Generates the information regarding the availability of credits
- . Updates the credit availability information depending upon the rdl_ufc_credits, rdl_ufc_type, rdl_ufc_valid (credit information from RDL) and credit information of transaction under process
- Does not update the credit information in case of nullified packet
- Informs Reordering Logic and FC-Checker about available credits

5.6.1.4 Capture Logic

- Capture Logic is pointer manager which manages the read and write pointers for the 4DW Buffer as per information extracted by Info-decoder

*5.6.1.5 ERR MSG TLP Generator**

- Generates Error Message TLP based upon the error message signals arriving from User Logic or RTL.

- Contains a configurable buffer to store error message signals

*5.6.1.6 Reordering Logi*c

- Forwards request for the VC Arbiter to enable the corresponding VC, if request asserted by user logic is forwarded from info-decoder when FC checker has passed the available credits check request asserted by ERR MSG TLP Generator or RTL
- Generates acknowledge for the requests arriving from RTL, user logic or error message TLP generator block
- Generates enable signal for MUX1 for selecting transaction from 4DW buffer, ERR MSG TLP generator or RTL
- Generates ttl_tlp_sop (start of packet) and ttl_tlp_ rdy (TL ready for transmission) signal for TDL

5.6.1.7 EOP Gen/Hold

- Generates eop (end of packet) signal for the error message TLP, and Config-completion TLPs
- Manages holding of eop signal from the user logic for virtual channel 0 only

5.6.1.8 Boosting Buffer

- . Used to avoid the large series of combo gating

5.6.1.9 MUX1

- Selects transaction from the ERR MSG TLP generator, 4DW buffer or Config-completion/UR (Unsupported request) from RTL

5.6.1.10 VC Arbiter

- . Generates select line for MUX2 signal using round robin and high priority algorithm
- . Acknowledges reordering logic of one of the Virtual Channels

5.6.1.11 ECRC Calculator

- Calculates 32-bit ECRC for the 64-bit TLP slots
- Appends the ECRC to the TLP during the transmission of the last 64-bit slot of the TLP.

- In the last TLP slot transfer if only 32-bits are valid then ECRC is appended within the same slot otherwise in the next one where only 32-bit CRC bits are declared as valid

5.6.1.12 ECRC Buffer

- Maintains TLP stream while appending the 32-bit ECRC with TLP
- Provides buffered output from TTL

5.6.1.13 Arbiter Holder

- Holds switching of the arbiter until entire TLP is transferred

5.6.1.14 MUX2

- Selects transaction from the one of the eight virtual channels

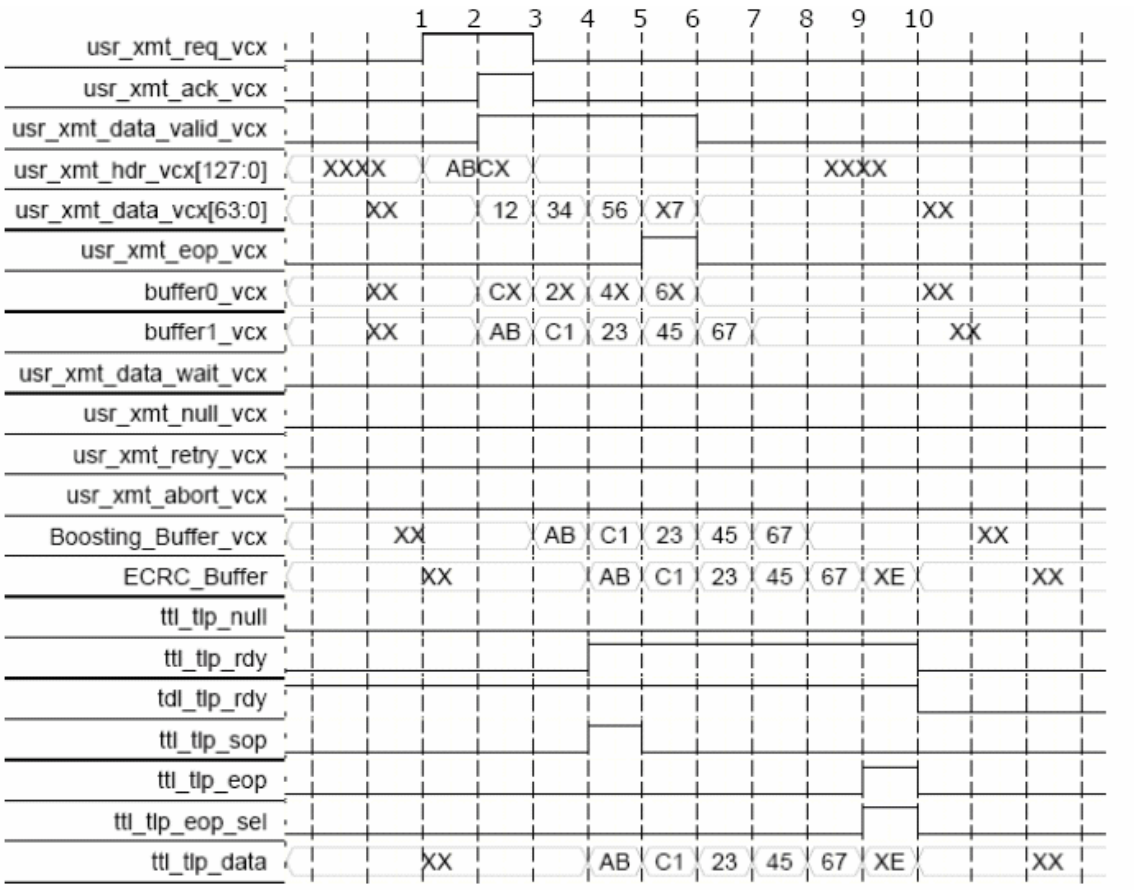


Fig 5-6 TLP with 3DW Hdr, DW aligned data, 1DW data valid in last slot Packet transmission from TTL to TDL

5.7 Design for Receive Transaction Layer (RTL)

The primary function of Receive Transaction Layer (RTL) block is to receive TLPs from Receive Data Link Layer (RDL) and pass them on to the User Logic. RTL block performs a series of checks (malformed TLP, UR or Configuration Request) on TLP received from RDL before passing it to user logic. Completions generated for Configuration Request and UR are passed on to the TTL block for transmission.

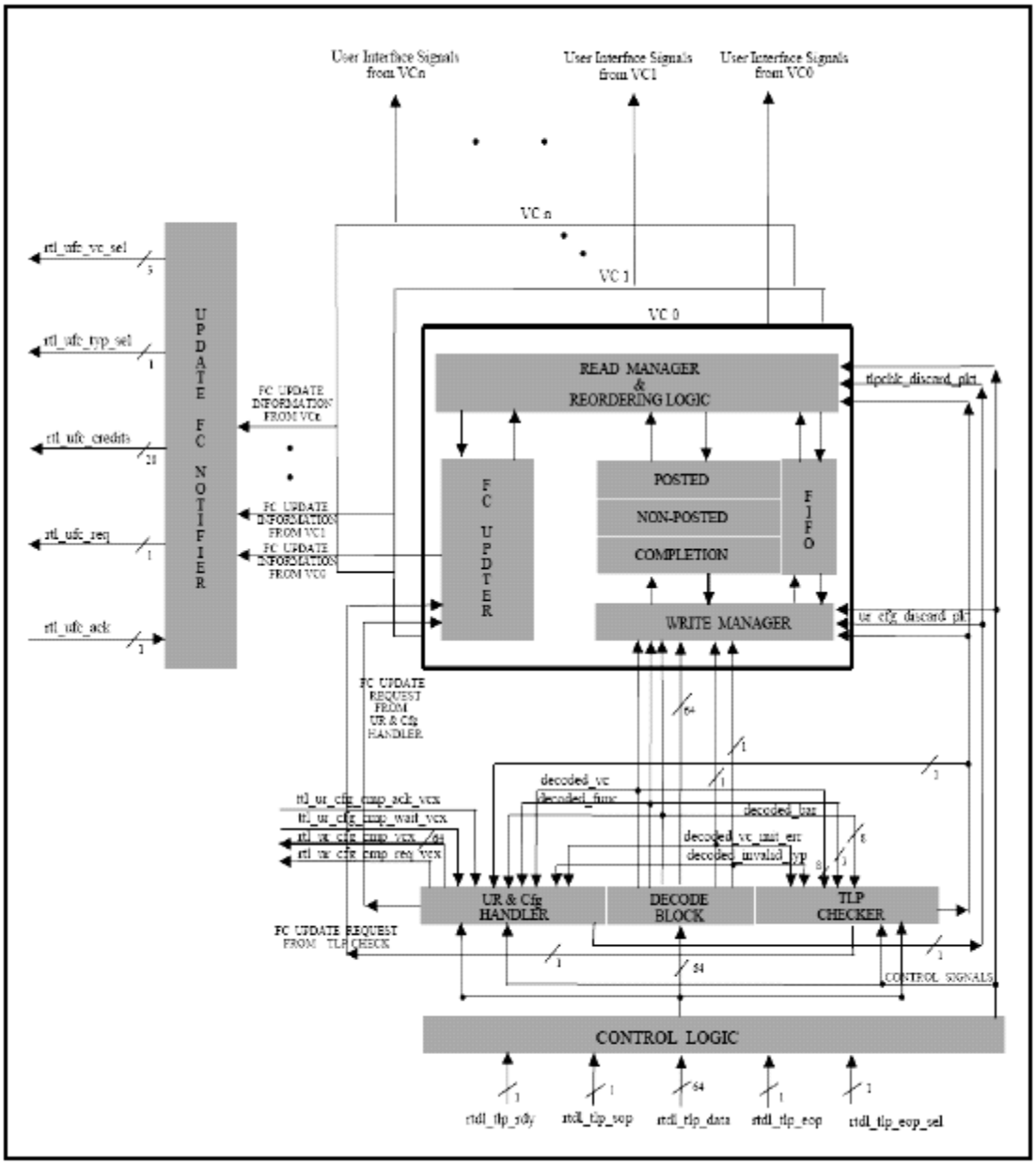


Fig 5-7 Block Diagram of Receive Transaction Layer

5.7.1 Description

5.7.1 Control Logic:

This logical sub-block generates control signals like start of packet, end of packet, etc for other sub-blocks of RTL thus ensures synchronization between all sub-blocks.

5.7.2 Decode Block:

This sub-block decodes the header of received TLP and makes the following information available to other sub-blocks:

- BAR decoding information using decoded_bar[7:0] signal
- . Function decoding information using decoded_func[2:0] signal. This information is required by User Logic along with BAR decoding information to complete the address decoding information.
- VC decoding information using decoded_vc[7:0] signal. One bit from this set of signals goes to the corresponding VC thus enabling the VC, which is intended recipient of TLP.
- . When a TLP is received with a TC mapped to an uninitialized VC this sub-block asserts its decoded_vc_init_err signal. Other sub-blocks if find this signal asserted for a received TLP, discard the TLP and credits are not updated for the same.
- . When a TLP is received with invalid Fmt -Type combination, this sub-block asserts its decoded_inv_typ signal. Other sub-blocks if find this signal asserted for a received TLP, discard the TLP and credits are not updated for the same.

5.7.3 TLP Checker:

This sub-block checks the TLP being received from RDL and reports other sub-blocks for an erroneous TLP.

- . This sub-block checks for a malformed TLP
- . ECRC check is also made by this block if TD bit is set in received TLP header
- . For an error in received TLP, this sub-block:
- . asserts signal tlp_chk_discard_pkt to indicate that the received TLP is to be discarded
- . informs FC updater of corresponding VC to update credits
- . request TTL to send an error message

- . FC update is performed only when no error is reported by RDL and Decode sub-block

5.7.4 UR and Cfg Handler:

This sub-block handles configuration and unsupported requests.

- . For configuration requests this sub-block:
- performs the corresponding read/write in configuration space
- generates completion and requests TTL for its transmission
- requests FC updater of VC0 to updates np credits once complete TLP is received, is not malformed, no error (like LCRC error) is reported by RDL and finally TTL accepts the request for transmitting completion
- . Request to TTL for transmitting completions and read/write in configuration space is made only when complete TLP is received, is not malformed and no error (like LCRC error) is reported by RDL
- . For poisoned configuration write a completion is sent with UR status
- . Configuration requests targeting user defined configuration space are not handled by this sub-block and are passed to User Logic.
- For unsupported requests this sub-block:
    - . generates completion with UR status for np requests and requests TTL for its transmission,
    - . requests FC updater of corresponding VC to update credits once complete TLP is received, is not malformed, no error (like LCRC error) is reported by RDL and TTL accepts the request for transmitting completion in case of np request.
- requests TTL for sending an error message
- . Request to TTL for transmitting completions and error message is made only when complete TLP is received, is not malformed and no error (like LCRC error) is reported by RDL.
- . This sub-block also make some additional checks like unsupported completion type, unexpected completion (Requester ID mismatch).
- . Data and handshake signals for transferring configuration and UR c ompletions to TTL are separate for each VC. This is required to maintain independent flow of

traffic through each VC.

5.7.5 Write Manager:

This sub-block manages storing of received TLPs into receive buffer.

- Each VC has a dedicated receive buffer and thus dedicated write/read managers and FC updater as shown in block diagram.
- Receive buffer of a VC is further divided into three different parts one each for posted requests, non-posted requests and completions. This makes retry and masking for a particular type of transaction from user logic easier.
- Write manager stores TLP only if the corresponding VC is intended recipient of the TLP and there is no error in the TLP.
- An error may be reported in the TLP by RDL or TLP Checker sub-block when the packet is being stored. In such a case write manager discards whatever was stored in buffer and restores the write pointer.

5.7.6 Read Manager and Reordering Logic:

This sub-block reorders and transfers TLPs from receive buffer to User Logic.

- Each VC has a dedicated read manager as explained earlier
- For transferring a TLP to User Logic, Read Manager:
- places header information on us r_rcv_hdr[127:0]
- asserts usr_rcv_req to request User Logic for TLP transfer
- places data on usr_rcv_data[63:0]
- places BAR hit information on usr_rcv_bar_hit[7:0]
- places function hit information on usr_rcv_func_hit[2:0]
- Read Manager also takes care of QW aligned address while placing data on data bus usr_rcv_data[63:0]. If Addr[2] bit is one i.e. address is not QW aligned,

read manager places first dword of data to be transferred on usr_rcv_data[63:32] and contents of usr_rcv_data[31:0] are don't care

- Read Manager is also responsible for making byte enable information available to User Logic while transferring data
- When a complete TLP is transferred to User Logic Read Manager requests FC updater of corresponding VC to update credits. Read Manager is also responsible

for reordering of TLPs when a retry is given or a particular type of TLP is halted for time being by User Logic

- Reordering of TLP is done according to PCI Express transaction ordering rules

5.7.7 FC Updater:

- On every update it triggers Update FC Notifier sub-block to send a request to TDL for transmitting Update FC DLLP.
- Credits are updated:
- when a malformed TLP is received with a valid Fmt -Type combination as reported by TLP Check sub-block
- when a complete TLP is transferred to User Logic as reported by Read Manager sub-block
- when a transmission of configuration request completion is started by TTL as requested by UR and Cfg Handler sub-block
- when transmission of an unsupported request completion is started by TTL as requested by UR and Configuration Handler sub-block
- Credits are not updated:
- when a TLP is received with a TC mapped to uninitialized VC as reported
- by Decode sub-block

For a TLP with invalid Fmt, Type combination as reported by decode sub block

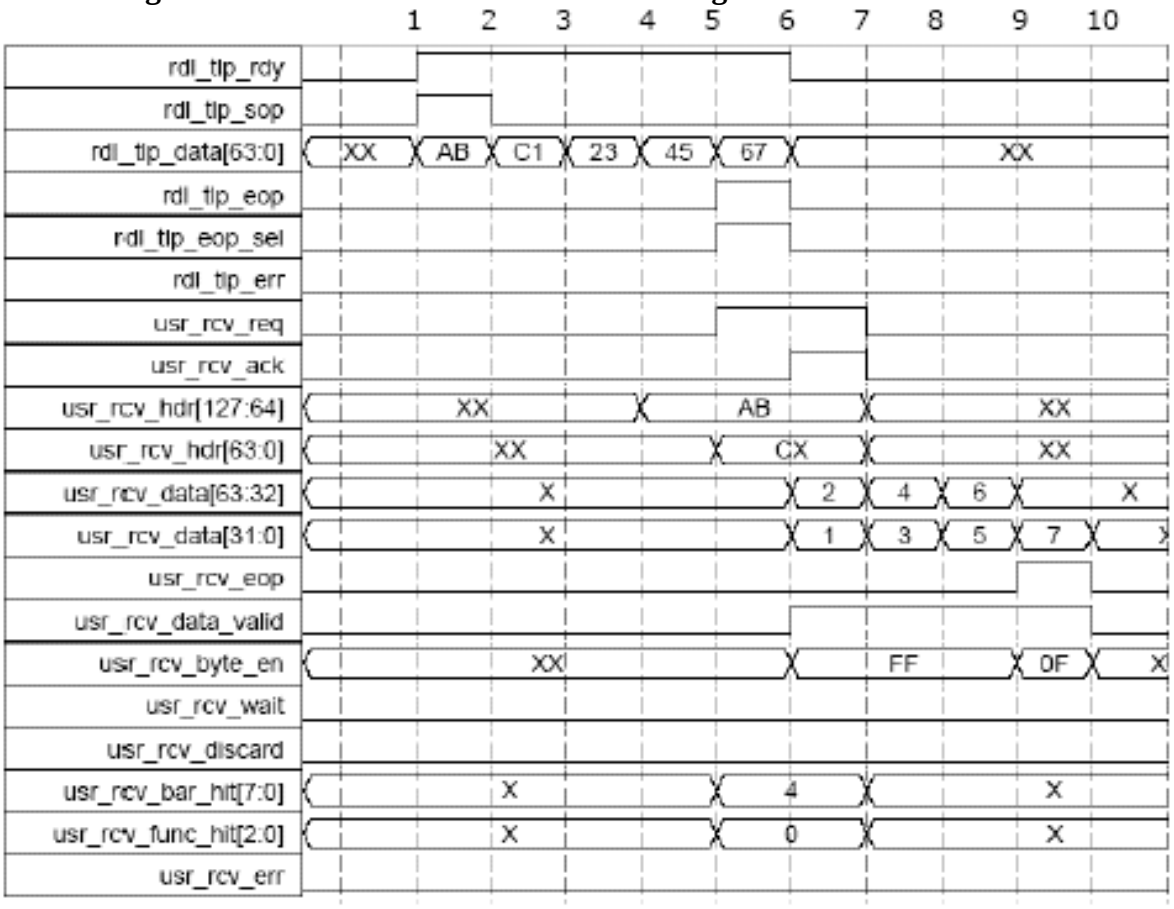
- when a received TLP causes the receive buffer overflow
- Credit information related to completion buffer space is made available to TTL by this sub-block. TTL if finds insufficient completion credits gives retry for a TLP transmit request from User Logic.

5.7.8 Update FC Notifier:

- This sub-block request TDL for sending Update FC DLLP.
- FC Updater sub-blocks of each VC report to this sub-block for any credit update.
- This sub-block then selects any one of those VC's and sends a request to TDL indicating that an update DLLP transmission is scheduled.
- Following information is made available to TDL for sending update DLLP:

- VC whose update credit is scheduled to be transmitted using signal rtl_ufc_vc_sel[2:0]
- updated credit value using signal rtl_ufc_credits[19:0]
- credit type: posted, non-posted using signal rtl_ufc_typ_sel
- Request to TDL for transmitting update FC is made by asserting signal rtl_ufc_req
- TDL acknowledges the request by asserting signal tdl_ufc_ack

Fig 5 -8 TLP transfer from RDL to User Logic without Error



Assumptions:

- 3 DW header , TLP includes data
- Address is QW aligned , X4 link established
- TLP is well formed and can be passed to User Logic

Fig 5-9 TLP transfer from RDL to User Logic without Error

Clocks	Event Description
1	• RDL asserts its start of packet rdl_tlp_sop and RDL ready rdl_tlp_rdy signals to indicate start of TLP transfer. • RDL also places first 2 DWs of TLP on rdl_tlp_data[63:0].
2, 3, 4	• RDL keeps rdl_tlp_rdy to indicate that data is valid and should be latched by RTL. • RDL places next 2 DWs on every clock. • RTL captures TLP data on every clock, checks it and stores it in its internal buffers.
5	• RDL places last 2 DWs of TLP on data bus rdl_tlp_data[63:0] and asserts end of packet signal rdl_tlp_eop to indicate last

In the above waveform RDL passes the packet to RTL and that is forwarded by RTL to user logic.

5.8 Verification Environment:

Before going into the results .Let us know about the total environment how different modules are connected and how the transactions are fired and how outputs are displayed

We have two ends where we are interested into check the outputs.

- One is ABFM(user logic) interface .We have got ABFM-Monitor which displays the values on the Interface bus .
- Second one is the Interface between the device connected between the PCI-Express core and the device connected .Here we connected an BFM(Bus Function Model) of PCI-Express which is configured as RC(root –complex).Here there is another Monitor printing the values on the differential signals .This prints all the training ordered sets used for link negotiations.

What Is a BFM ?

A BFM simply models the bus interface of some unit, regularly a microprocessor. It does not contain the RTL or gate level specifics of the unit's internal workings. The purpose of a BFM is to gain simulation speed, ease of use, and ease of creation.

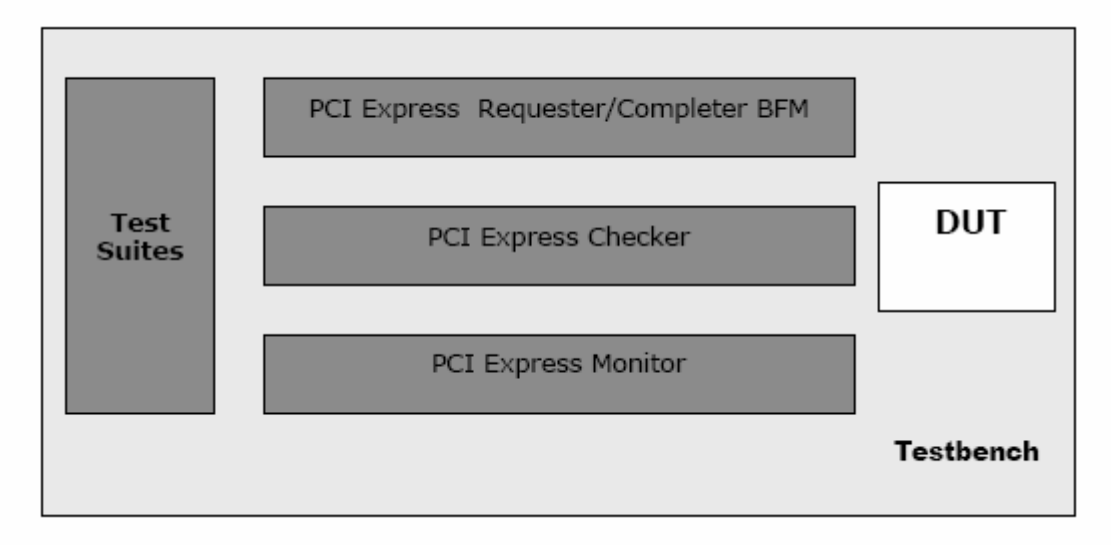


Fig 5-10 Verification Environment

Here DUT includes the PCI-Express core which includes Physical-layer to Application BFM

5.8.1 What is Verification?

Verification is not synonymous with simulation. It is a strategy to make sure all aspects of the system meets the specification document, while simulation is a tool used in the verification effort. The basic components of verification are shown in "The Verification Pyramid" below .

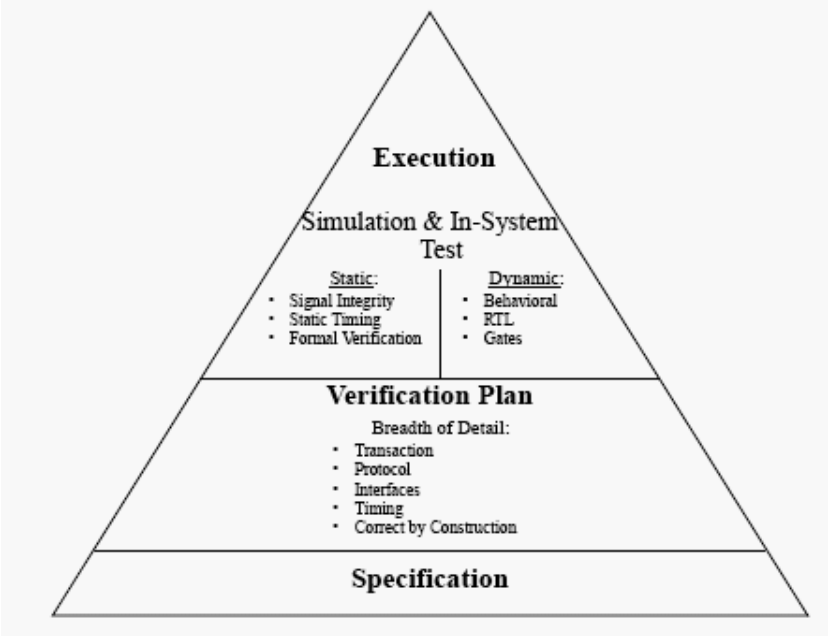


Fig 5-11 Verification Pyramid

Test Scenario or Application Layer

This layer is the highest level of interaction with the DUT for test suite creation. At this level, users will have control over creating a combination of different test scenarios based on the specific application. Through setting parameters either in the constraint segments of the class or parameters for the sequence generation, pseudo-random test suites can be generated. Checking the results is of course more domains specific..

Application Layer here is designed as ABFM.

5.9 Results and Observations :

Here we present the output of the Design through the two monitors tracking on both ends of the PCI-Express core .

As shown in the ABFM Output (**Refer to APPENDIX for Output Log file**s) Memory and IO transactions are transmitted from ABFM. As we can see the output obtained in the log files written by Monitors and Protocol Checkers Present at both ends of the PCI-Express Core check for violations.

In the physical layer the packet is serialized and then they are transmitted on to the Links that are detected during the Link training .Link training is the process of negotiating the link, lane width and data transmission rate. This is done by symbol encoding at the PHY interface, there are special symbols called Training ordered sets used for training process.

From BFM-OUTPUT It can be seen that Link training is done by transmitting the ordered sets namely **TS1, TS2, SKIP, IDLE**. These out puts are displayed by BFM-Monitor present on the link. It can also be observed that all the packets transmitted from the ABFM are transmitted on the link differentially.

Chapter 6
SUMMARY AND CONCLUSION

The work in this thesis presents an approach to design Transaction Layer of PCI-Express and to verify it by the design of configurable Application Layer named here as Application Bus Function Model (ABFM). The design for the Transaction Layer implemented here produces one of the optimized cost-effective structures for the design of PCI-Express core.

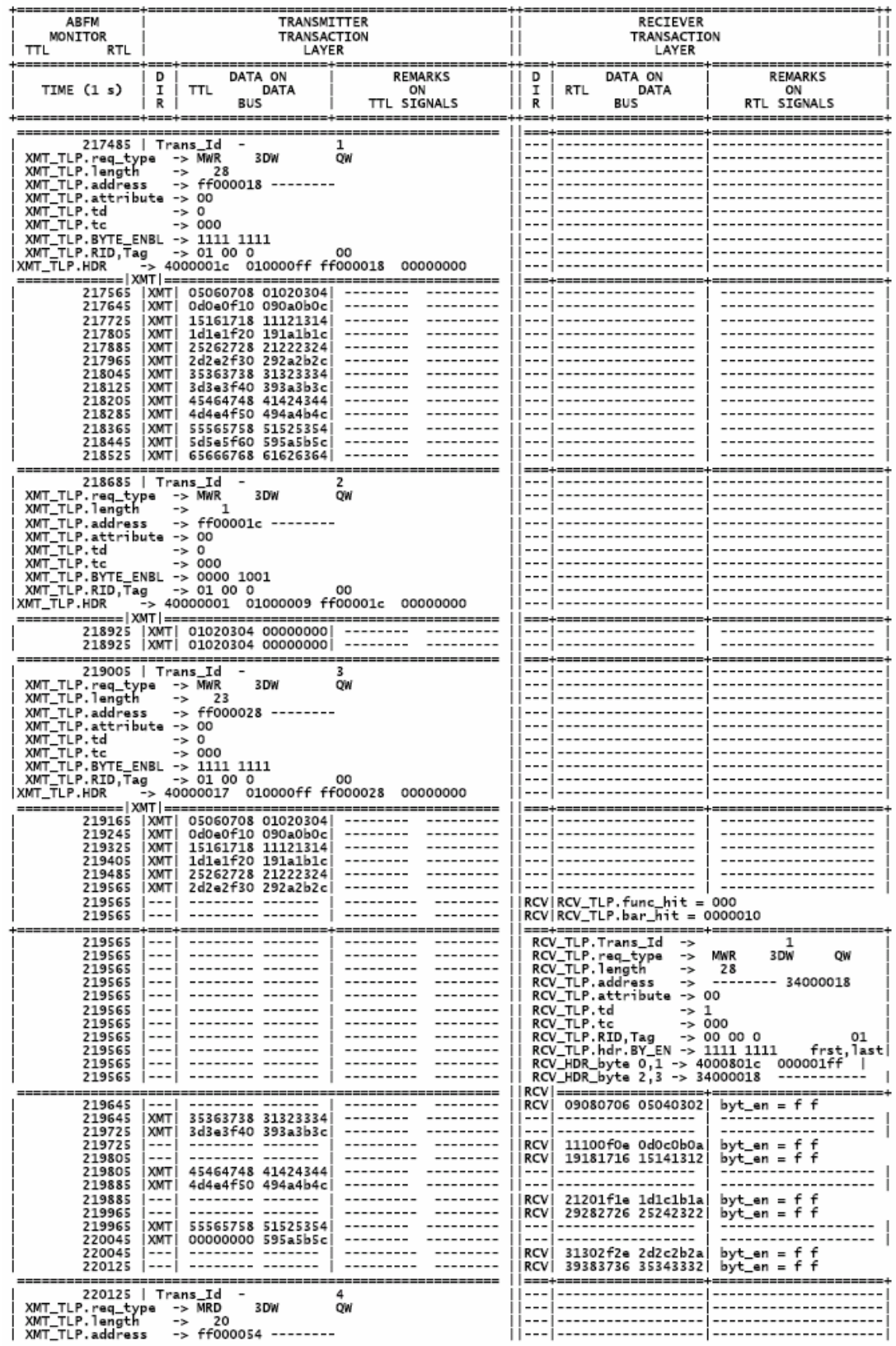
PCI Express provides many benefits to the workstation designer and to workstation customers. These benefits result from the architecture, the implementation and standardization.. Also, because the low pin count of the links, workstation designers have much more flexibility in designing electromechanical layouts, potentially lowering costs and increasing serviceability. PCI Express will serve as a general purpose I/O interconnects for a wide variety of future computing and communications platforms. Gradually PCI Express will replace PCI and PCI-X as the core I/O subsystem for workstations4. In the meantime, customers will gain immediate benefit from the features of PCI Express while protecting their current investments in I/O devices, software and operating environments.

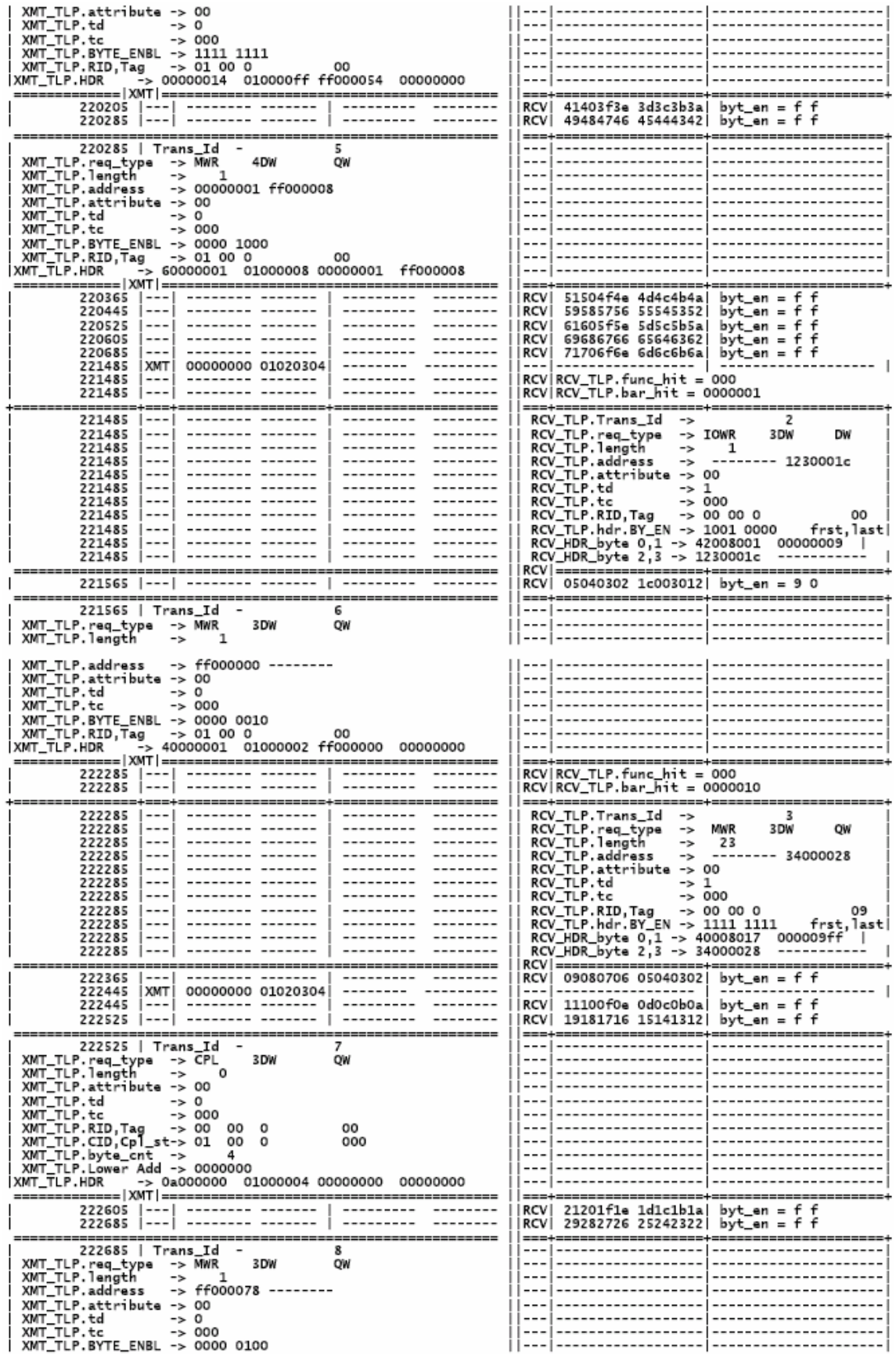
FUTURE SCOPE :

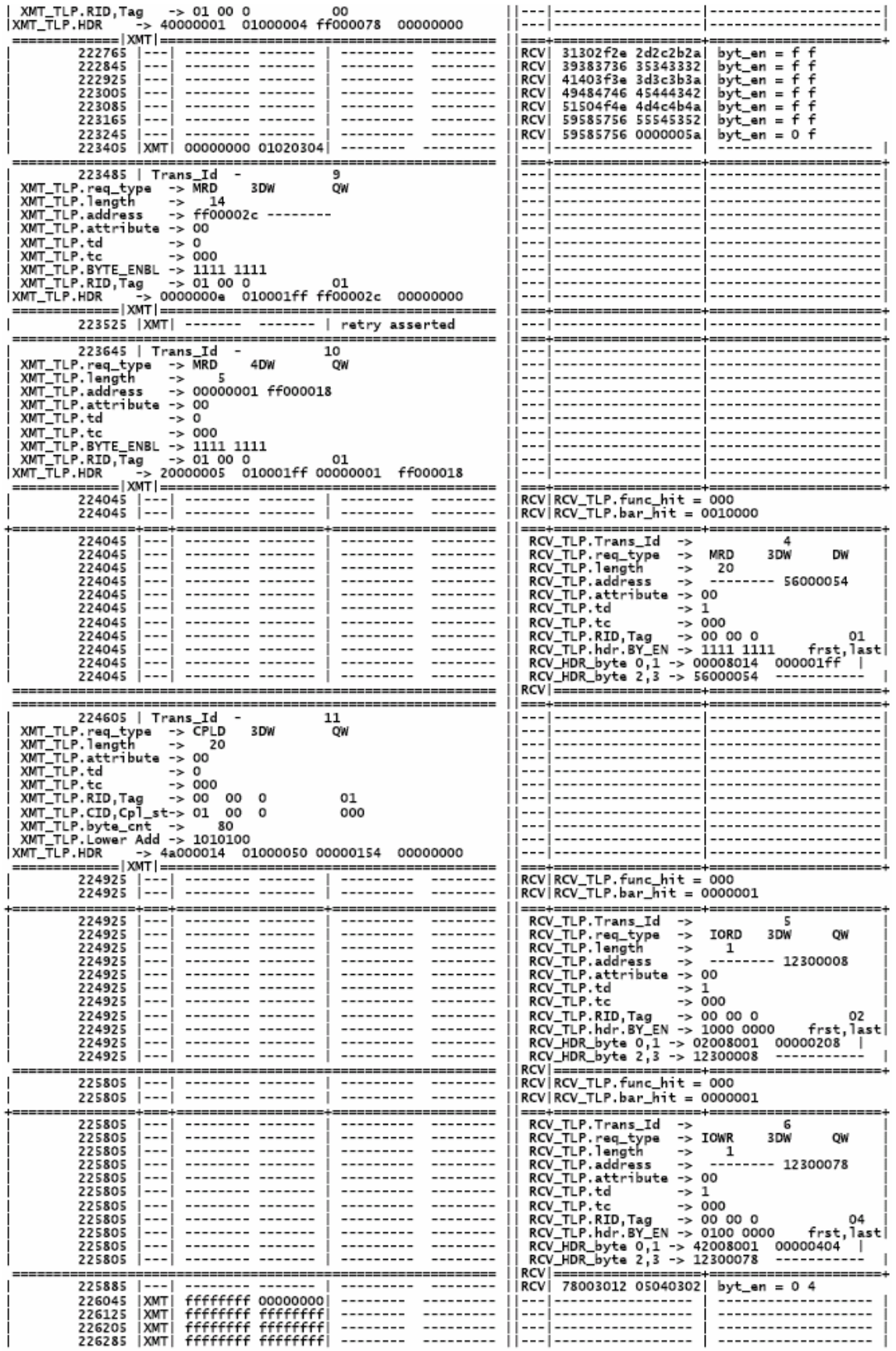
PCI Express will serve as a general purpose I/O interconnect for a wide variety of future computing and communications platforms.The approach for the design of Transaction Layer described in this thesis is synthesized on X1,X2 and X4 Further it can be extended and can be synthesized on X16,X32 . This thesis work uses PCI-Express base 1.0 a as guide which is generation one of PCI-Express ,the design described can further extended and can be implemented following PCI-Express Base 2.0 which is generation two of PCI-Express. PCI-Express generation two can transfer data still more faster at 5 Giga bits/sec as compared to 2.5 Giga bits/sec of generation one.

Verification PCI-Express at 2.5 Gigabits/sec had been a very expensive task .The approach used in this work is formal verification .The performance of the design can be further enhanced by testing the design using Enhanced Loop back technique.

APPENDIX : ABFM-MONITOR OUTPUT




XMT_TLP.attribute -> 00
XMT_TLP.td -> 0
XMT_TLP.tc -> 000
XMT_TLP.BYTE_ENBL -> 1111 1111
XMT_TLP.RID,Tag -> 01 00 0 00
XMT_TLP.HDR -> 00000014 010000ff ff000054 00000000
XMT
220205 --- -------- -------- | ---------- ---------
220285 --- -------- -------- | ---------- ---------
RCV| 41403f3e 3d3c3b3a| byt_en = f f
RCV| 49484746 45444342| byt_en = f f
220285 | Trans_Id - 5
XMT_TLP.req_type -> MWR 4DW QW
XMT_TLP.length -> 1
XMT_TLP.address -> 00000001 ff000008
XMT_TLP.attribute -> 00
XMT_TLP.td -> 0
XMT_TLP.tc -> 000
XMT_TLP.BYTE_ENBL -> 0000 1000
XMT_TLP.RID,Tag -> 01 00 0 00
XMT_TLP.HDR -> 60000001 01000008 00000001 ff000008
220365 --- -------- -------- | ---------- ---------
220445 --- -------- -------- | ---------- ---------
220525 --- -------- -------- | ---------- ---------
220605 --- -------- -------- | ---------- ---------
220685 --- -------- -------- | ---------- ---------
221485 XMT 00000000 01020304 -------- ----------
221485 --- -------- -------- | ---------- ---------
221485 --- -------- -------- | ---------- ---------
RCV| 51504f4e 4d4c4b4a| byt_en = f f
RCV| 59585756 55545352| byt_en = f f
RCV| 61605f5e 5d5c5b5a| byt_en = f f
RCV| 69686766 65646362| byt_en = f f
RCV| 71706f6e 6d6c6b6a| byt_en = f f
RCV|RCV_TLP.func_hit = 000
RCV|RCV_TLP.bar_hit = 0000001
221485 --- -------- -------- | ---------- ---------
221565 --- -------- -------- | ---------- ---------
RCV_TLP.Trans_Id -> 2
RCV_TLP.req_type -> IOWR 3DW DW
RCV_TLP.length -> 1
RCV_TLP.address -> -------- 1230001c
RCV_TLP.attribute -> 00
RCV_TLP.td -> 1
RCV_TLP.tc -> 000
RCV_TLP.RID,Tag -> 00 00 0 00
RCV_TLP.hdr.BY_EN -> 1001 0000 frst,last
RCV_HDR_byte 0,1 -> 42008001 00000009
RCV_HDR_byte 2,3 -> 1230001c ----------
RCV
RCV| 05040302 1c003012| byt_en = 9 0
221565 | Trans_Id - 6
XMT_TLP.req_type -> MWR 3DW QW
XMT_TLP.length -> 1
XMT_TLP.address -> ff000000 --------
XMT_TLP.attribute -> 00
XMT_TLP.td -> 0
XMT_TLP.tc -> 000
XMT_TLP.BYTE_ENBL -> 0000 0010
XMT_TLP.RID,Tag -> 01 00 0 00
XMT_TLP.HDR -> 40000001 01000002 ff000000 00000000
222285 --- -------- -------- | ---------- ---------
222365 --- -------- -------- | ---------- ---------
222445 XMT 00000000 01020304 -------- ----------
222445 --- -------- -------- | ---------- ---------
222525 --- -------- -------- | ---------- ---------
RCV|RCV_TLP.func_hit = 000
RCV|RCV_TLP.bar_hit = 0000010
222285 --- -------- -------- | ---------- ---------
RCV_TLP.Trans_Id -> 3
RCV_TLP.req_type -> MWR 3DW QW
RCV_TLP.length -> 23
RCV_TLP.address -> -------- 34000028
RCV_TLP.attribute -> 00
RCV_TLP.td -> 1
RCV_TLP.tc -> 000
RCV_TLP.RID,Tag -> 00 00 0 09
RCV_TLP.hdr.BY_EN -> 1111 1111 frst,last
RCV_HDR_byte 0,1 -> 40008017 000009ff
RCV_HDR_byte 2,3 -> 34000028 ----------
RCV
RCV| 09080706 05040302| byt_en = f f
RCV| 11100f0e 0d0c0b0a| byt_en = f f
RCV| 19181716 15141312| byt_en = f f
222525 | Trans_Id - 7
XMT_TLP.req_type -> CPL 3DW QW
XMT_TLP.length -> 0
XMT_TLP.attribute -> 00
XMT_TLP.td -> 0
XMT_TLP.tc -> 000
XMT_TLP.RID,Tag -> 00 00 0 00
XMT_TLP.CID,Cpl_st-> 01 00 0 000
XMT_TLP.byte_cnt -> 4
XMT_TLP.Lower Add -> 0000000
XMT_TLP.HDR -> 0a000000 01000004 00000000 00000000
226085 --- -------- -------- | ---------- ---------
226885 --- -------- -------- | ---------- ---------
RCV| 21201f1e 1d1c1b1a| byt_en = f f
RCV| 29282726 25242322| byt_en = f f
226885 | Trans_Id - 8
XMT_TLP.req_type -> MWR 3DW QW
XMT_TLP.length -> 1
XMT_TLP.address -> ff000078 --------
XMT_TLP.attribute -> 00
XMT_TLP.td -> 0
XMT_TLP.tc -> 000
XMT_TLP.BYTE_ENBL -> 0000 0100


XMT_TLP.RID,Tag -> 01 00 0 00
XMT_TLP.HDR -> 40000001 01000004 ff000078 00000000
XMT
222765 --- -------- -------- -------- --------
222845 --- -------- -------- -------- --------
222925 --- -------- -------- -------- --------
223005 --- -------- -------- -------- --------
223085 --- -------- -------- -------- --------
223165 --- -------- -------- -------- --------
223245 --- -------- -------- -------- --------
223405 XMT 00000000 01020304 -------- --------
223485 Trans_Id - 9
XMT_TLP.req_type -> MRD 3DW QW
XMT_TLP.length -> 14
XMT_TLP.address -> ff00002c --------
XMT_TLP.attribute -> 00
XMT_TLP.td -> 0
XMT_TLP.tc -> 000
XMT_TLP.BYTE_ENBL -> 1111 1111
XMT_TLP.RID,Tag -> 01 00 0 01
XMT_TLP.HDR -> 0000000e 010001ff ff00002c 00000000
XMT
223525 XMT ------- ------- retry asserted
223645 Trans_Id - 10
XMT_TLP.req_type -> MRD 4DW QW
XMT_TLP.length -> 5
XMT_TLP.address -> 00000001 ff000018
XMT_TLP.attribute -> 00
XMT_TLP.td -> 0
XMT_TLP.tc -> 000
XMT_TLP.BYTE_ENBL -> 1111 1111
XMT_TLP.RID,Tag -> 01 00 0 01
XMT_TLP.HDR -> 20000005 010001ff 00000001 ff000018
XMT
224045 --- -------- -------- -------- --------
224045 --- -------- -------- -------- --------
224045 --- -------- -------- -------- --------
224045 --- -------- -------- -------- --------
224045 --- -------- -------- -------- --------
224045 --- -------- -------- -------- --------
224045 --- -------- -------- -------- --------
224045 --- -------- -------- -------- --------
224045 --- -------- -------- -------- --------
224045 --- -------- -------- -------- --------
224605 Trans_Id - 11
XMT_TLP.req_type -> CPLD 3DW QW
XMT_TLP.length -> 20
XMT_TLP.attribute -> 00
XMT_TLP.td -> 0
XMT_TLP.tc -> 000
XMT_TLP.RID,Tag -> 00 00 0 01
XMT_TLP.CID,Cpl_st-> 01 00 0 000
XMT_TLP.byte_cnt -> 80
XMT_TLP.Lower Add -> 1010100
XMT_TLP.HDR -> 4a000014 01000050 00000154 00000000
XMT
224925 --- -------- -------- -------- --------
224925 --- -------- -------- -------- --------
224925 --- -------- -------- -------- --------
224925 --- -------- -------- -------- --------
224925 --- -------- -------- -------- --------
224925 --- -------- -------- -------- --------
224925 --- -------- -------- -------- --------
224925 --- -------- -------- -------- --------
224925 --- -------- -------- -------- --------
224925 --- -------- -------- -------- --------
225805 --- -------- -------- -------- --------
225805 --- -------- -------- -------- --------
225805 --- -------- -------- -------- --------
225805 --- -------- -------- -------- --------
225805 --- -------- -------- -------- --------
225805 --- -------- -------- -------- --------
225805 --- -------- -------- -------- --------
225805 --- -------- -------- -------- --------
225805 --- -------- -------- -------- --------
225805 --- -------- -------- -------- --------
225885 --- -------- -------- -------- --------
226045 XMT ffffffff 00000000 -------- --------
226125 XMT ffffffff ffffffff -------- --------
226205 XMT ffffffff ffffffff -------- --------
226285 XMT ffffffff ffffffff -------- --------
RCV 31302f2e 2d2c2b2a byt_en = f f
RCV 39383736 35343332 byt_en = f f
RCV 41403f3e 3d3c3b3a byt_en = f f
RCV 49484746 45444342 byt_en = f f
RCV 51504f4e 4d4c4b4a byt_en = f f
RCV 59585756 55545352 byt_en = f f
RCV 59585756 0000005a byt_en = 0 f
RCV RCV_TLP.func_hit = 000
RCV RCV_TLP.bar_hit = 0010000
RCV_TLP.Trans_Id -> 4
RCV_TLP.req_type -> MRD 3DW DW
RCV_TLP.length -> 20
RCV_TLP.address -> -------- 56000054
RCV_TLP.attribute -> 00
RCV_TLP.td -> 1
RCV_TLP.tc -> 000
RCV_TLP.RID,Tag -> 00 00 0 01
RCV_TLP.hdr.BY_EN -> 1111 1111 frst,last
RCV_HDR_byte 0,1 -> 00008014 000001ff
RCV_HDR_byte 2,3 -> 56000054 ----------
RCV
RCV RCV_TLP.func_hit = 000
RCV RCV_TLP.bar_hit = 0000001
RCV_TLP.Trans_Id -> 5
RCV_TLP.req_type -> IORD 3DW QW
RCV_TLP.length -> 1
RCV_TLP.address -> -------- 12300008
RCV_TLP.attribute -> 00
RCV_TLP.td -> 1
RCV_TLP.tc -> 000
RCV_TLP.RID,Tag -> 00 00 0 02
RCV_TLP.hdr.BY_EN -> 1000 0000 frst,last
RCV_HDR_byte 0,1 -> 02008001 00000208
RCV_HDR_byte 2,3 -> 12300008 ----------
RCV
RCV RCV_TLP.func_hit = 000
RCV RCV_TLP.bar_hit = 0000001
RCV_TLP.Trans_Id -> 6
RCV_TLP.req_type -> IOWR 3DW QW
RCV_TLP.length -> 1
RCV_TLP.address -> -------- 12300078
RCV_TLP.attribute -> 00
RCV_TLP.td -> 1
RCV_TLP.tc -> 000
RCV_TLP.RID,Tag -> 00 00 0 04
RCV_TLP.hdr.BY_EN -> 0100 0000 frst,last
RCV_HDR_byte 0,1 -> 42008001 00000404
RCV_HDR_byte 2,3 -> 12300078 ----------
RCV
RCV 78003012 05040302 byt_en = 0 4

BFM output

```
                D     T   SYMBOL  F CNT BYTE CNT                          F L      CID   MSG
                I SEQ A     /     M /    / HI      LO          PACKET     B B       /     /    T E
  TIME (100 ps) R  #  G   PACKET  T LEN ADDRESS  ADDRESS    HI        LO   E E RID  VID  STATUS D P   ECRC     LCRC
-- Training - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
                Started
  4880 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  5520 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  6000 D --- -- Training - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
                Started
  6000 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  6160 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  6640 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  6800 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  7280 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  7440 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  7920 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  8080 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  8560 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  8720 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  9200 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  9360 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
  9840 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 10000 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 10480 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 10640 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 11120 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 11280 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 11760 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 11920 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 12400 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 12560 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 13040 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 13200 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 13680 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 13840 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 14320 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 14480 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 14960 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 15120 U --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 15600 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 15760 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 16240 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 16400 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 16880 D --- -- TS1      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 17040 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 17520 D --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 17680 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 18160 D --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 18320 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 18800 D --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 18960 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 19440 D --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 19600 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 20080 D --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 20240 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 20720 D --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 20880 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 21360 D --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 21520 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 22000 D --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 22160 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 22640 D --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 54320 U --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 54440          Log        *** Link is negotiated at X   4 ***
 54800 D --- -- TS2      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 54920          Log        *** Link is negotiated at X   4 ***
 56040 D --- -- IDLE     - 007 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 56040 D --- -- Training - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
                Done
 56040 D --- -- LO State - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 56080 D --- -- IFC1_P   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
 56160 D --- -- IFC1_NP  - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
 56240 D --- -- IFC1_CPL - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
 56320 D --- -- IFC1_P   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
 56400 D --- -- IFC1_NP  - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
 56480 D --- -- IFC1_CPL - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
 56520 U --- -- IDLE     - 013 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
 56520 U --- -- Training - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
                Done
```

```
56520 U --- -- LO State - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
56560 U --- -- IFC1_P   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
56560 D --- -- IFC1_P   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
56640 U --- -- IFC1_NP  - --- -------- -------- -------- -------- - - ---- ---- ------ 0  7f 200 - -
56640 D --- -- IFC1_NP  - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
56720 U --- -- IFC1_CPL - --- -------- -------- -------- -------- - - ---- ---- ------ 0  00 000 - -
56720 D --- -- IFC1_CPL - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
56800 U --- -- IFC1_P   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
56800 D --- -- IFC1_P   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
56880 U --- -- IFC1_NP  - --- -------- -------- -------- -------- - - ---- ---- ------ 0  7f 200 - -
56880 D --- -- IFC1_NP  - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
56960 U --- -- IFC1_CPL - --- -------- -------- -------- -------- - - ---- ---- ------ 0  00 000 - -
56960 D --- -- IFC1_CPL - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
57040 U --- -- IFC1_P   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
57040 D --- -- IFC1_P   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
57120 U --- -- IFC1_NP  - --- -------- -------- -------- -------- - - ---- ---- ------ 0  7f 200 - -
57120 D --- -- IFC1_NP  - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
57200 U --- -- IFC1_CPL - --- -------- -------- -------- -------- - - ---- ---- ------ 0  00 000 - -
57200 D --- -- IFC1_CPL - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -

59440 D 000 -- MSGD     3 001 -------- -------- 00000450 74008001 - - 0000 ---- SSPLS  4  01 001 1 0
                                               00000000 00010000
                                               -------- 00000000
59520 U --- -- IFC2_NP  - --- -------- -------- -------- -------- - - ---- ---- ------ 0  7f 200 - -
59600 U --- -- IFC2_CPL - --- -------- -------- -------- -------- - - ---- ---- ------ 0  00 000 - -
59680 U --- -- IFC2_P   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  20 200 - -
59760 U --- -- IFC2_NP  - --- -------- -------- -------- -------- - - ---- ---- ------ 0  7f 200 - -
59840 U --- -- IFC2_CPL - --- -------- -------- -------- -------- - - ---- ---- ------ 0  00 000 - -
59920 U --- -- IFC2_CPL - --- -------- -------- -------- -------- - - ---- ---- ------ 0  00 000 - -
60760 U --- -- IDLE     - 020 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
60800 U 000 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
61360 D --- -- IDLE     - 047 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
61600 D 001 00 CFG_WR0  2 001 -------- 01000010 0000000f 44008001 f 0 0000 ---- ------ -- 01 001 1 0
                                               -------- 01000010
                                               -------- ffffffff
63000 U --- -- IDLE     - 054 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
63040 U 001 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
63480 U --- -- IDLE     - 010 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
63640 U 000 00 CPL      0 000 004      00       01000004 0a000000 - - 0000 0100 SC     -- 01 --- 0 0
                                               -------- 00000000
63960 U --- -- IDLE     - 007 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
64000 U --- -- UFC_NP   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  80 201 - -
64240 D --- -- IDLE     - 065 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
64440 D 002 00 CFG_RD0  0 001 -------- 01000010 0000000f 04008001 f 0 0000 ---- ------ -- 01 --- 1 0
                                               -------- 01000010
65800 U --- -- IDLE     - 044 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
65840 U 002 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
66280 U --- -- IDLE     - 010 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
66480 U 001 00 CPLD     2 001 004      00       01000004 4a000001 - - 0000 0100 SC     -- 01 001 0 0
                                               -------- 00000000
                                               -------- 0100ffff
66760 U --- -- IDLE     - 006 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
66800 U --- -- UFC_NP   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  81 201 - -
67000 D --- -- IDLE     - 063 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
67240 D 003 00 CFG_WR0  2 001 -------- 01000010 0000000f 44008001 f 0 0000 ---- ------ -- 01 001 1 0
                                               -------- 01000010
                                               -------- 00003012
67480 D --- -- IDLE     - 005 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
67520 D 001 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
68600 U --- -- IDLE     - 044 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
68640 U 003 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
69080 U --- -- IDLE     - 010 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
69240 U 002 00 CPL      0 000 004      00       01000004 0a000000 - - 0000 0100 SC     -- 01 --- 0 0
                                               -------- 00000000
69560 U --- -- IDLE     - 007 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
69600 U --- -- UFC_NP   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  82 202 - -
69840 D --- -- IDLE     - 057 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
69880 D 002 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
70120 D 004 00 CFG_RD0  0 001 -------- 01000010 0000000f 04008001 f 0 0000 ---- ------ -- 01 --- 1 0
                                               -------- 01000010

71480 U --- -- IDLE     - 046 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
71520 U 004 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
71960 U --- -- IDLE     - 010 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
72160 U 003 00 CPLD     2 001 004      00       01000004 4a000001 - - 0000 0100 SC     -- 01 001 0 0
                                               -------- 00000000
                                               -------- 01003012
72440 U --- -- IDLE     - 006 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
72480 U --- -- UFC_NP   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  83 202 - -
72680 D --- -- IDLE     - 063 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
72720 D 003 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
80720 D --- -- IDLE     - 199 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
80960 D 005 00 CFG_WR0  2 001 -------- 01000014 0000000f 44008001 f 0 0000 ---- ------ -- 01 001 1 0
                                               -------- 01000014
                                               -------- ffffffff
```

```
114920 U 00d 01 CPL      0 000 004      00       01000004 0a000000 - - 0000 0100 SC     -- 01 --- 0 0
                                                 -------- 00000100
115000 D 00f 00 CFG_RD0  0 001 -------- 01000018 0000000f 04008001 f 0 0000 ---- ------ -- 01 --- 1 0
                                                 -------- 01000018
115240 U --- -- IDLE     - 007 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
115280 U --- -- UFC_NP   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  8d 208 - -
115480 D --- -- IDLE     - 011 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
115520 D 00d -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
115760 D 010 01 CFG_RD0  0 001 -------- 0100001c 0000010f 04008001 f 0 0000 ---- ------ -- 01 --- 1 0
                                                 -------- 0100001c
116360 U --- -- IDLE     - 026 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
116400 U 00f -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
116840 U --- -- IDLE     - 010 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
117040 U 00e 00 CPLD     2 001 004      00       01000004 4a000001 - - 0000 0100 SC     -- 01 001 0 0
                                                 -------- 00000000
                                                 -------- 04000089
117320 U --- -- IDLE     - 006 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
117360 U 010 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
117440 U --- -- UFC_NP   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  8e 208 - -
117600 D --- -- IDLE     - 045 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
117640 D 00e -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
117720 U --- -- IDLE     - 006 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
117920 U 00f 01 CPLD     2 001 004      00       01000004 4a000001 - - 0000 0100 SC     -- 01 001 0 0
                                                 -------- 00000100
                                                 -------- 07000000
219280 D 021 01 MWR      2 01c -------- 34000018 000001ff 4000801c f f 0000 ---- ------ -- 01 007 1 0
                                                 -------- 34000018
                                                 06070809 02030405
                                                 0e0f1011 0a0b0c0d
                                                 16171819 12131415
                                                 1e1f2021 1a1b1c1d
                                                 26272829 22232425
                                                 2e2f3031 2a2b2c2d
                                                 36373839 32333435
                                                 3e3f4041 3a3b3c3d
                                                 46474849 42434445
                                                 4e4f5051 4a4b4c4d
                                                 56575859 52535455
                                                 5e5f6061 5a5b5c5d
                                                 66676869 62636465
                                                 6e6f7071 6a6b6c6d
219520 D --- -- IDLE     - 005 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
219720 U 020 00 MWR      2 01c -------- ff000018 010000ff 4000001c f f 0100 ---- ------ -- 01 007 0 0
                                                 -------- ff000018
                                                 08070605 04030201
                                                 100f0e0d 0c0b0a09
                                                 18171615 14131211
                                                 201f1e1d 1c1b1a19
                                                 28272625 24232221
                                                 302f2e2d 2c2b2a29
                                                 38373635 34333231
                                                 403f3e3d 3c3b3a39
                                                 48474645 44434241
                                                 504f4e4d 4c4b4a49
                                                 58575655 54535251
                                                 605f5e5d 5c5b5a59
                                                 68676665 64636261
                                                 706f6e6d 6c6b6a69
219760 D 022 00 IO_WR    2 001 -------- 1230001c 00000009 42008001 9 0 0000 ---- ------ -- 01 001 1 0
                                                 -------- 1230001c
                                                 -------- 02030405
219800 U --- -- IDLE     - 001 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
220000 U 021 00 MWR      2 001 -------- ff00001c 01000009 40000001 9 0 0100 ---- ------ -- 01 001 0 0
                                                 -------- ff00001c
                                                 -------- 04030201
220000 D --- -- IDLE     - 005 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
221120 U 022 00 MWR      2 017 -------- ff000028 010000ff 40000017 f f 0100 ---- ------ -- 01 006 0 0
                                                 -------- ff000028
                                                 08070605 04030201
                                                 100f0e0d 0c0b0a09
                                                 18171615 14131211
                                                 201f1e1d 1c1b1a19
                                                 28272625 24232221
                                                 302f2e2d 2c2b2a29
                                                 38373635 34333231
                                                 403f3e3d 3c3b3a39
                                                 48474645 44434241
                                                 504f4e4d 4c4b4a49
                                                 58575655 54535251
                                                 -------- 5c5b5a59
221120 D 023 09 MWR      2 017 -------- 34000028 000009ff 40008017 f f 0000 ---- ------ -- 01 006 1 0
                                                 -------- 34000028
                                                 06070809 02030405
                                                 0e0f1011 0a0b0c0d
                                                 16171819 12131415
                                                 1e1f2021 1a1b1c1d
                                                 26272829 22232425
                                                 2e2f3031 2a2b2c2d
                                                 36373839 32333435
                                                 3e3f4041 3a3b3c3d
                                                 46474849 42434445
                                                 4e4f5051 4a4b4c4d
                                                 56575859 52535455
                                                 -------- 5a000000
221360 D --- -- IDLE     - 005 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
221400 D 020 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
221400 U --- -- IDLE     - 006 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
221440 U 022 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
221480 D 021 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
221640 U --- -- IDLE     - 004 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
221720 D 024 01 MRD      0 014 -------- 56000054 000001ff 00008014 f f 0000 ---- ------ -- 01 --- 1 0
                                                 -------- 56000054
221800 D --- -- IDLE     - 001 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
```

```
221800 U 023 00 MRD      0 014 -------- ff000054 010000ff 00000014 f f 0100 ---- ------ -- 01 --- 0 0
                                                 -------- ff000054
221840 D 022 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
222080 D 025 02 IO_RD    0 001 -------- 12300008 00000208 02008001 8 0 0000 ---- ------ -- 01 --- 1 0
                                                 -------- 12300008
222120 U --- -- IDLE     - 007 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
222160 U --- -- UFC_P    - --- -------- -------- -------- -------- - - ---- ---- ------ 0  21 207 - -
222240 D --- -- IDLE     - 003 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
222360 U --- -- IDLE     - 004 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
222440 D 026 03 CFG_RD0  0 001 -------- 01000000 00000302 04008001 2 0 0000 ---- ------ -- 01 --- 1 0
                                                 -------- 01000000
222600 U 024 00 MWR      3 001 00000001 ff000008 01000008 60000001 8 0 0100 ---- ------ -- 01 001 0 0
                                                 ff000008 00000001
                                                 -------- 04030201
222680 D --- -- IDLE     - 005 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
222720 D 023 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
222920 U --- -- IDLE     - 007 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
222960 U 023 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
223000 D 027 04 IO_WR    2 001 -------- 12300078 00000404 42008001 4 0 0000 ---- ------ -- 01 001 1 0
                                                 -------- 12300078
                                                 -------- 02030405
223040 U --- -- UFC_NP   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  a0 211 - -
223120 U 024 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
223160 D --- -- IDLE     - 003 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
223200 D 024 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
223320 U --- -- IDLE     - 004 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
223440 D 028 05 MRD      0 00e -------- 3400002c 000005ff 0000800e f f 0000 ---- ------ -- 01 --- 1 0
                                                 -------- 3400002c
223520 U 025 00 MWR      2 001 -------- ff000000 01000002 40000001 2 0 0100 ---- ------ -- 01 001 0 0
                                                 -------- ff000000
                                                 -------- 04030201
223600 D --- -- IDLE     - 003 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
223800 U --- -- IDLE     - 006 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
223840 U 026 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
223840 D 029 06 MRD      1 005 00000007 89000018 000006ff 20008005 f f 0000 ---- ------ -- 01 --- 1 0
                                                 89000018 00000007
224040 U --- -- IDLE     - 004 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
224080 D --- -- IDLE     - 005 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
224120 D 025 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
224200 U 026 00 CPL      0 000 004      00       01000004 0a000000 - - 0000 0100 SC     -- 01 --- 0 0
                                                 -------- 00000000
224280 U --- -- IDLE     - 001 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
224400 D 02a 07 MRD      1 003 00000007 89000020 000007ff 20008003 f f 0000 ---- ------ -- 01 --- 1 0
                                                 89000020 00000007
224480 U 027 00 MWR      2 001 -------- ff000078 01000004 40000001 4 0 0100 ---- ------ -- 01 001 0 0
                                                 -------- ff000078
                                                 -------- 04030201
224480 D --- -- IDLE     - 001 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
224720 D 02b 08 MRD      1 005 00000007 8900001c 000008ff 20008005 f f 0000 ---- ------ -- 01 --- 1 0
                                                 8900001c 00000007
224760 U --- -- IDLE     - 006 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
224800 U 028 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
224880 U --- -- UFC_P    - --- -------- -------- -------- -------- - - ---- ---- ------ 0  22 20d - -
224960 D --- -- IDLE     - 005 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
225000 D 026 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
225160 U --- -- IDLE     - 006 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
225280 D 02c 09 MRD      1 012 00000007 89000044 000009ff 20008012 f f 0000 ---- ------ -- 01 --- 1 0
                                                 89000044 00000007
225360 U 028 03 CPLD     2 001 004      00       01000004 4a000001 - - 0000 0100 SC     -- 01 001 0 0
                                                 -------- 00000300
                                                 -------- af180000
225360 D --- -- IDLE     - 001 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
225400 D 027 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
225640 U --- -- IDLE     - 006 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
225680 U 029 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
225760 U --- -- UFC_NP   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  a2 211 - -
225840 U 02a -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
226040 U --- -- IDLE     - 004 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
226240 U 029 01 MRD      1 005 00000001 ff000018 010001ff 20000005 f f 0100 ---- ------ -- 01 --- 0 0
                                                 ff000018 00000001
226520 U --- -- IDLE     - 006 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
226560 U 02b -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
226640 U --- -- UFC_NP   - --- -------- -------- -------- -------- - - ---- ---- ------ 0  a3 211 - -
226720 U 02c -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
226920 U --- -- IDLE     - 004 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
226960 D 02d 03 MWR      3 020 00000007 8900003c 000003ff 60008020 f f 0000 ---- ------ -- 01 008 1 0
                                                 8900003c 00000007
                                                 06070809 02030405
                                                 0e0f1011 0a0b0c0d
                                                 16171819 12131415
                                                 1e1f2021 1a1b1c1d
                                                 26272829 22232425
                                                 2e2f3031 2a2b2c2d
                                                 36373839 32333435
                                                 3e3f4041 3a3b3c3d
                                                 46474849 42434445
                                                 4e4f5051 4a4b4c4d
                                                 56575859 52535455
                                                 5e5f6061 5a5b5c5d
                                                 66676869 62636465
                                                 6e6f7071 6a6b6c6d
                                                 76777879 72737475
                                                 7e7f8000 7a7b7c7d
227120 D --- -- IDLE     - 003 -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
227160 D 028 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
227240 D 029 -- ACK      - --- -------- -------- -------- -------- - - ---- ---- ------ -- -- --- - -
```

REFERENCES

[1] Eugin Hyun, Kwang-Su Seong, **"**The design of PCI Express for future communication platform**"**Intelligent Signal Processing and Communication Systems, 2004. ISPACS 2004. Proceedings of 2004 International Symposium on 18-19 Nov. 2004 Page(s):734 - 739

[2] Dhawan, S.K**"** Introduction to PCI Express - A New High Speed Serial Data Bus" Nuclear Science Symposium Conference Record, 2005 IEEE Oct 23 - 29, 2005 Page(s):687 - 691

[3] Min-An Song, Ting-Chun Huang, Sy-Yen Kuo "A Functional Verification Environment for Advanced Switching Architecture"
Electronic Design, Test and Applications, 2006. DELTA 2006. Third IEEE International Workshop on 17-19 Jan. 2006 Page(s):201 – 204

[4] Aguilar, M, Veloz, A, Guzman. M Proposal of implementation of the "data link layer" of PCI-express Electrical and Electronics Engineering, 2004. (ICEEE). 1st International Conference on 24-27 June 2004 Page(s):64 – 69

[5] Abdennadher, S.; Shaikh, S.A "Challenges in High Speed Interface Testing" Test Symposium, 2005. Proceedings. 14th Asian 18-21 Dec. 2005 Page(s):468 - 468

[6] Dawson,C Pattanam, S.K, Roberts. D **"** The Verilog Procedural Interface for the Verilog Hardware Description Language**"** Verilog HDL Conference, 1996. Proceedings., 1996 IEEE International 26-28 Feb. 1996 Page(s):17 – 23

[7] Szu-Tsung Cheng, Brayton, R.K. York, G. Yelick, K. Saldanha, A Compiling Verilog into timed finite state machines Verilog HDL Conference, 1995. Proceedings., 1995 IEEE International 27-29 March 1995 Page(s):32 – 39

[8] PCI-Express Base specifications 1.0a PCI-SIG

[9] Plessier.B, Pixley.C.” Formal verification of a commercial serial bus interface**”** Computers and Communications, 1995. Conference Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on 28-31 Mar 95 Page(s):378 – 382

[10] A samir Palnitkar “Verilog –HDL tutorial”