Design and Implementation of FPGA Based P.C.I. / P.C.I. Express Bridge

B. Tech Project 1st Stage Report

Submitted in partial fulfillment of the requirements
for the degree of

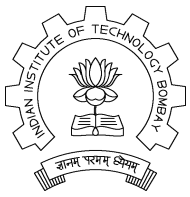
Bachelor of Technology

by

Sushil Chauhan
Roll No: 01005002

under the guidance of

Prof. S.S.S.P. Rao



Department of Computer Science and Engineering
Indian Institute of Technology
Bombay
November 16, 2004

Acknowledgement

November 16, 2004

I would like to express my sincere gratitude to my guide Prof. S.S.S.P. Rao, for his constant support, encouragement and direction without which the realization of this Project would not have been possible. His friendly disposition and invaluable support and encouragement has proved to be very rewarding.

Sushil Chauhan
01005002

Abstract

This B-Tech. Project is aimed at the Design and Implementation of an FPGA - based P.C.I. / P.C.I. Express Bridge. In order to achieve this goal, we need to have the complete knowledge of P.C.I. and P.C.I. Express Specifications. This First Stage Report is aimed to cover the Fundamentals of the P.C.I. Express I/O Interconnect. The report presents an introduction to the P.C.I. Express and its benefits. It presents a detailed overview of the P.C.I. Express Layered Architecture and the Credits-based FLow Control Mechanism followed by P.C.I. Express. I have also given the Proposed Design Outline for the Implementation of FPGA - based P.C.I. / P.C.I. Express Bridge.

Contents

List of Figures

Chapter 1

Introduction

There are certain times in the evolution of technology that serve as inflection points that forever change the course of events. For the computing sector and communications, the adoption of P.C.I. Express will serve as one of these inflection points. P.C.I. stands for Peripheral Component Interconnect. P.C.I. Express is the high-performance interconnect that gives "more for less", meaning more bandwidth with fewer pins. P.C.I. Express is a high speed, low voltage, differential serial pathway for two devices to communicate with each other. P.C.I. Express uses a protocol that allows devices to communicate simultaneously by implementing dual unidirectional paths between two devices. P.C.I. Express provides a speed of 2.5 Gigabits per second per direction. The theoritical bandwidth of P.C.I. Express nearly doubles the theoritical bandwidth of P.C.I. with approximately one tenth the number of pins [1].

Aside from connecting two devices, P.C.I. Express provides two methods to satisfy the growing bandwidth or data-throughput capacity requirements between two devices. The P.C.I. Express can be scaled in a linear manner by adding additional serial lanes that work together to meet the bandwidth objectives of a device. A key advantage of P.C.I. Express scalability is that it can be scaled on a device-to-device basis within a system of multiple P.C.I. Express connections. For instance, if one device-to-device interconnect requires P.C.I. Express to be scaled to meet its bandwidth objectives, the rest of the P.C.I. Express system device-to-device interconnections are not affected.

1.1 Benefits of P.C.I. Express

P.C.I. Express provides benefits across five main vectors: layered architecture, high performance, I/O simplification, next generation multimedia and ease of use.

1.1.1 Layered Architecture

The P.C.I. Express layered architecture improves serviceability and scalability. The Software Layer maintains software compatiability with P.C.I. to ease migration. In brief, the three layers which form the core of P.C.I. Express are Transaction Layer, Data Link Layer and Physical Layer. The layered architecture is critical to enabling the scalability of P.C.I. Express. For the next generation speed bump from 2.5 gigabits per second to probably more than 5.0 gigabits per second, only the Physical Layer needs to evolve.

1.1.2 High Performance

The bus efficiency is determined by several factors such as protocol and design limitations and is beyond the scope of this report. We are using Maximum theoritical bandwidth to compare various applications. For example, P.C.I. is a 32-bit bus running at 33 MHz, so it can provide

the Maximum bandwidth of 132 Megabytes per second. P.C.I.-X is a 64-bit bus running at 66 MHz for a total of 533 Megabytes per second. But approximately, 85 percent of computing industry continues to use the 33-MHz version. However, the P.C.I. bus cannot actually transfer data at these rates due to overhead required for commands as well as inability to perform reads and writes at the same time.

The initial implementations of P.C.I. Express utilize a 2.5 gigabits per second per direction bandwidth but the capability of the bus has the potential for growth to 10 gigabits per second per direction. P.C.I. Express provides the bandwidth scalability of 250 megabytes per second per direction for initial single lane (Lane is defined in the next chapter) to 32000 megabytes per second per direction for 10 gigabits per second per direction signaling across 32 lanes.

1.1.3 I/O Simplification

In today's computing platforms, there is a lot of overabudance of I/O technologies. Platforms have P.C.I.-X for servers, Cardbus on mobile PCs and P.C.I. for desktop PCs. P.C.I. Express provides a unique interface technology serving multiple market segments. For example, a PC chipset designer may implement a x16 P.C.I. Express configuration for graphics, a x1 configuration for general purpose I/O and a x4 configuration as a high speed chip-to-chip interconnects. (Different P.C.I. Express Link configurations are defined in next chapter)

1.1.4 Next Generation Multimedia

P.C.I. Express provides new capabilities for multimedia not available in the platform today-namely Isochronous support. Isochronous is a specific type of QoS (Quality of Service) guarantee that data is delivered using a deterministic and time-dependent method.

1.1.5 Ease of Use

P.C.I. Express natively supports hot swap and hot plug. Hot swap is the ability to swap I/O cards wihtout software interaction where as hot plug may require operating system interaction. P.C.I. Express as a hardware specification defines the capability to support both hot swap and hot plug, but hot plug support depends on the operating system. In the future, systems will not need to be powered down to replace faulty equipment or install upgrades [1].

Chapter 2

PCI Express Architecture Overview

2.1 Links and Lanes

The connection between two PCI Express devices is called ***Link***. A link consists of a number of ***Lanes***. A lane is the term used for a single set of differential transmit (TX) and receive pairs (RX). The ***PCI Express Base Specification*** defines the following configuration of serial links: ***x1, x2, x4, x8, x12, x16 and x32*** [1]. For example, a ***x1*** configuration indicates that the link between two PCI Express devices consists of a single lane. A ***x4*** configuration indicates that the link between two PCI Express devices consists of 4 lanes. Following figures shows ***x1*** and ***x4*** links.

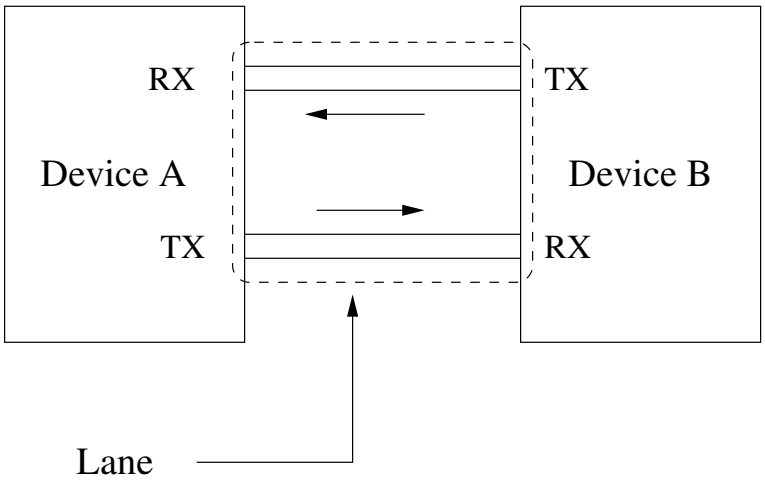


Figure 2.1: ***x1*** Link

At the device, the collection of transmitter and receiver pairs that are assosiated with a link is referred to as a ***Port***. The transmitter of one device must be the receiver for the other device. Like links, a device's port can be made up of multiple lanes.

2.1.1 Multiple Lanes

Much like lanes can be added to a highway to increase the total traffic throughput, multiple lanes can be used within a PCI Express link to increase the available bandwidth. As previously noted, the maximum bandwidth of a ***x1*** link is 250 megabytes per second per direction. Because PCI Express is dual unidirectional, this offers a maximum theoritical bandwidth of 500 megabytes per second between the two devices. For example, the ***x4*** link has a maximum bandwidth of 1000 megabytes per second per direction and the ***x16*** link has a maximum bandwidth of 4000

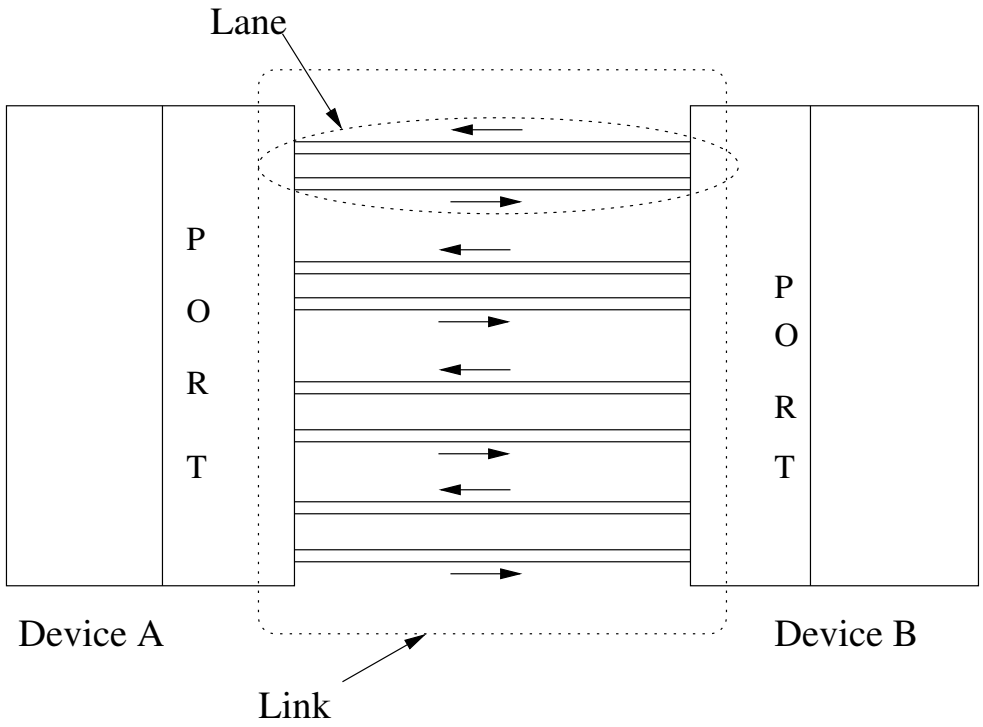


Figure 2.2: ***x4*** Link, Lanes and Ports

megabytes per second per direction. This means that PCI Express can be adjusted to meet a variety of applications with a variety of bandwidth needs.

2.2 Device Types

The ***PCI Express Base Specification*** [1] identifies 4 types of PCI Express Devices. These are :

- Root Complex
- PCI Express to PCI bridge
- Endpoint
- Switch

2.2.1 Root Complex

The Root complex is the head or root of the connection of the I/O system to the CPU and the Memory. Each interface off of the root complex defines a separate hierarchy domain.

2.2.2 PCI Express to PCI bridge

A PCI Express to PCI bridge has one PCI Express port and one or multiple PCI/PCI-X bus interfaces. This element allows PCI Express to co-exist on a platform with existing PCI

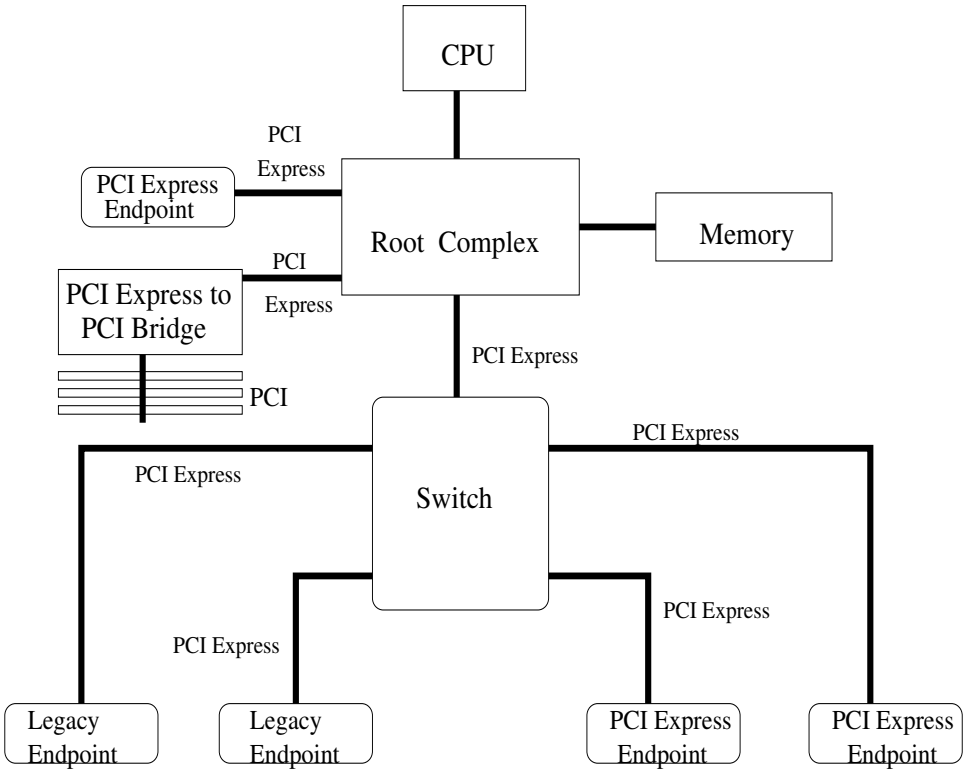


Figure 2.3: PCI Express Devices

technologies. This device must fully support all PCI and/or PCI-X transactions on its PCI interface(s). It must follow a variety of rules for properly transforming those PCI/PCI-X transactions into PCI Express transactions.

2.2.3 Endpoint

An Endpoint is a device that can request/complete PCI Express transactions for itself (for ex., a graphics device) or on behalf of a non-PCI Express device. There are two types of endpoints: legacy and PCI Express, and they are differentiated by the types of transactions they support.

2.2.4 Switch

A Switch is used to fan out a PCI Express hierarchy. It is responsible for properly forwarding transactions to the appropriate link. Unlike a root complex, it must always manage peer-to-peer transactions between downstream devices (downstream means the side further away from the root complex).

2.3 PCI Express Transactions

Transactions form the basis for the transformation of information between PCI Express devices. PCI Express uses a ***split-transaction*** protocol. This means that there are two portions to the transaction, the ***request*** and the ***completion***. The transaction initiator referred to as the ***requester***,sends out the request packet. It makes its way towards the intended target

of the request, referred to as the ***completer***. For requests that require completions, the completer later sends back a completion packet (or packets) to the requester. A completion is not necessarily required for each request.

Even though PCI Express links are point-to-point, this does not always mean that one of the devices on the link is the requester and the other the completer. For example, say that the root complex in Figure 2.3 wants to communicate with a PCI Express endpoint that is downstream of the switch. The root complex is the requester and the endpoint is the completer. Even though, the switch receives the transaction from the root complex, it is not considered a completer of that transaction. Even though, the endpoint receives the transaction from the switch, it does not consider the switch to be requester of that transaction. The requester identifies itself within the request packet it sends out, and this informs the completer where it should return the completion packets (if needed).

2.3.1 Transaction Types

The PCI Express architecture defines four transaction types: memory, I/O, configuration and message [1].

Memory Transactions

Transactions targeting the memory space transfer data to or from a memory-mapped location. There are several types of memory transactions: Memory Read Request, Memory Read Completion and Memory Write Request. Memory transactions use either 32-bit addressing or 64-bit addressing.

I/O Transactions

Transactions targeting the I/O space transfer data to or from an I/O mapped location. There are several types of I/O transactions: I/O Read Request, I/O Read Completion, I/O Write Request and I/O Write Completion. I/O transactions use only 32-bit addressing.

Configuration Transactions

Transactions targeting the configuration space are used for device configuration and setup. These transactions access the configuration registers of PCI Express devices. There are several types of configuration transactions: Configuration Read Request, Configuration Read Completion, Configuration Write Request and Configuration Write Completion.

Message Transactions

PCI Express adds this new transaction type to communicate a variety of miscellaneous messages between PCI Express devices. These transactions are used for functions like interrupt signaling, error signaling or power management. This transaction type is necessary since these functions are no longer available via sideband signals such as PME, SERR and so on.

2.4 Architectural Build Layers

The specification defines three abstract layers that build a PCI Express transaction [1]. These are:

- **Transaction Layer :** The main responsibility of this layer is to begin the process of turning requests or completion data from the device core into a PCI Express transaction.
- **Data Link Layer :** The main responsibility of this layer is to ensure that the transactions going back and forth across the link are received properly.
- **Physical Layer :** This layer is responsible for the actual transmitting and receiving of the transaction across the PCI express link.

Since each PCI Express link is dual uni-directional, each of these architectural layers has transmit as well as receive functions assosiated with it. Outgoing PCI Express transactions may proceed from the transmit side of the Transaction Layer to the transmit side of the Data Link Layer to the transmit side of the Physical Layer. Incoming transactions may proceed from the receive side of the Physical Layer to the receive side of the Data Link Layer and then to the transmit side of the Transaction Layer.

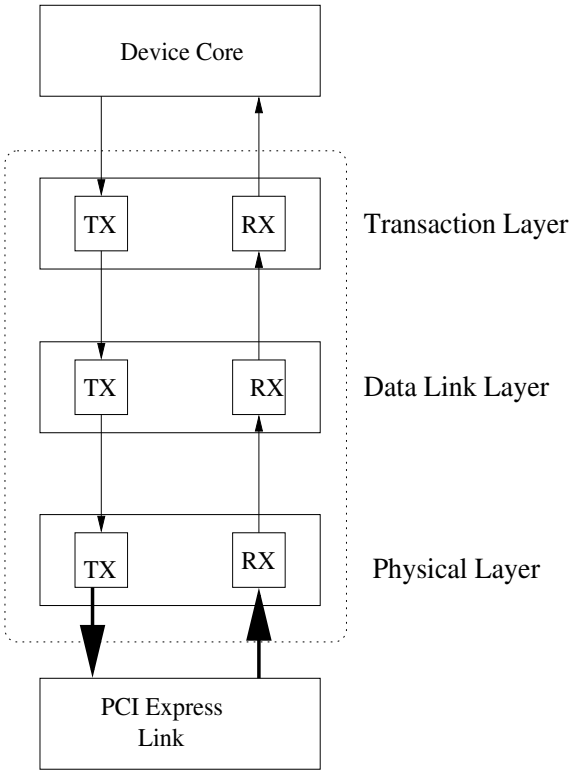


Figure 2.4: The Three Architectural Build Layers

2.5 Packet Formation

As a transaction flows through the transmitting PCI Express device, each architectural layer adds on its specific information [1]. The Transaction Layer generates a ***header*** and adds the ***data payload*** (if required) and an optional ***ECRC*** (end-to-end CRC). The Data Link Layer adds the ***sequence number*** and ***LCRC*** (link CRC). The Physical Layer ***frames*** it for proper transmission to the other device.

When it gets to the receiver side of the link mate, the complete reversal of this build occurs. The Physical Layer decodes the framing characters and passes along the rest of the information (sequence number, header, data, ECRC and LCRC) to its Data Link Layer. The Data Link Layer checks out the sequence number and LCRC, and then passes the header, data and ECRC on to the Transaction Layer. The Transaction Layer decodes the ECRC (if applicable) and header, and then passes the appropriate information on to its device core. Figure 2.5 shows the buildup of a Transaction Layer Packet through Architectural Layers of a PCI Express Device. And figure 2.6 shows the Flow of Transaction Layer Packet between two PCI Express devices [3].

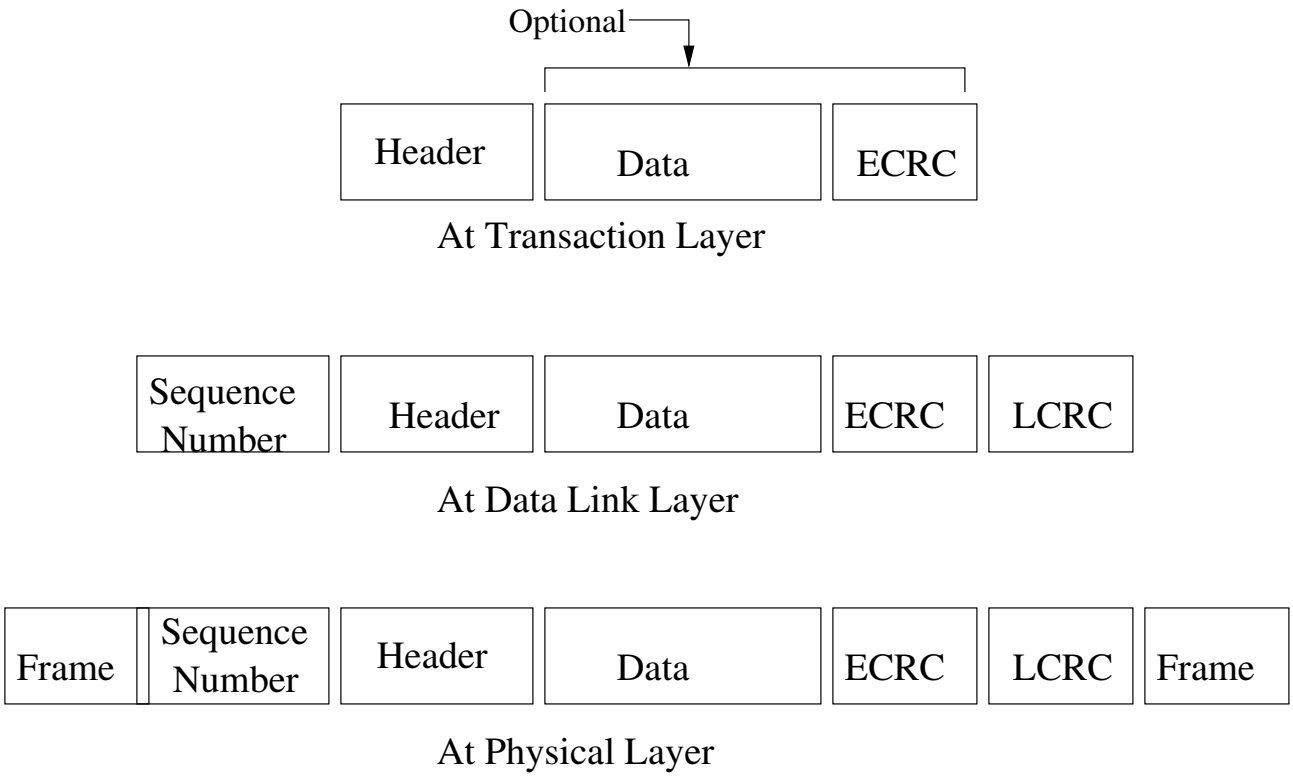


Figure 2.5: Transaction Buildup for a TLP through Architectural Layers

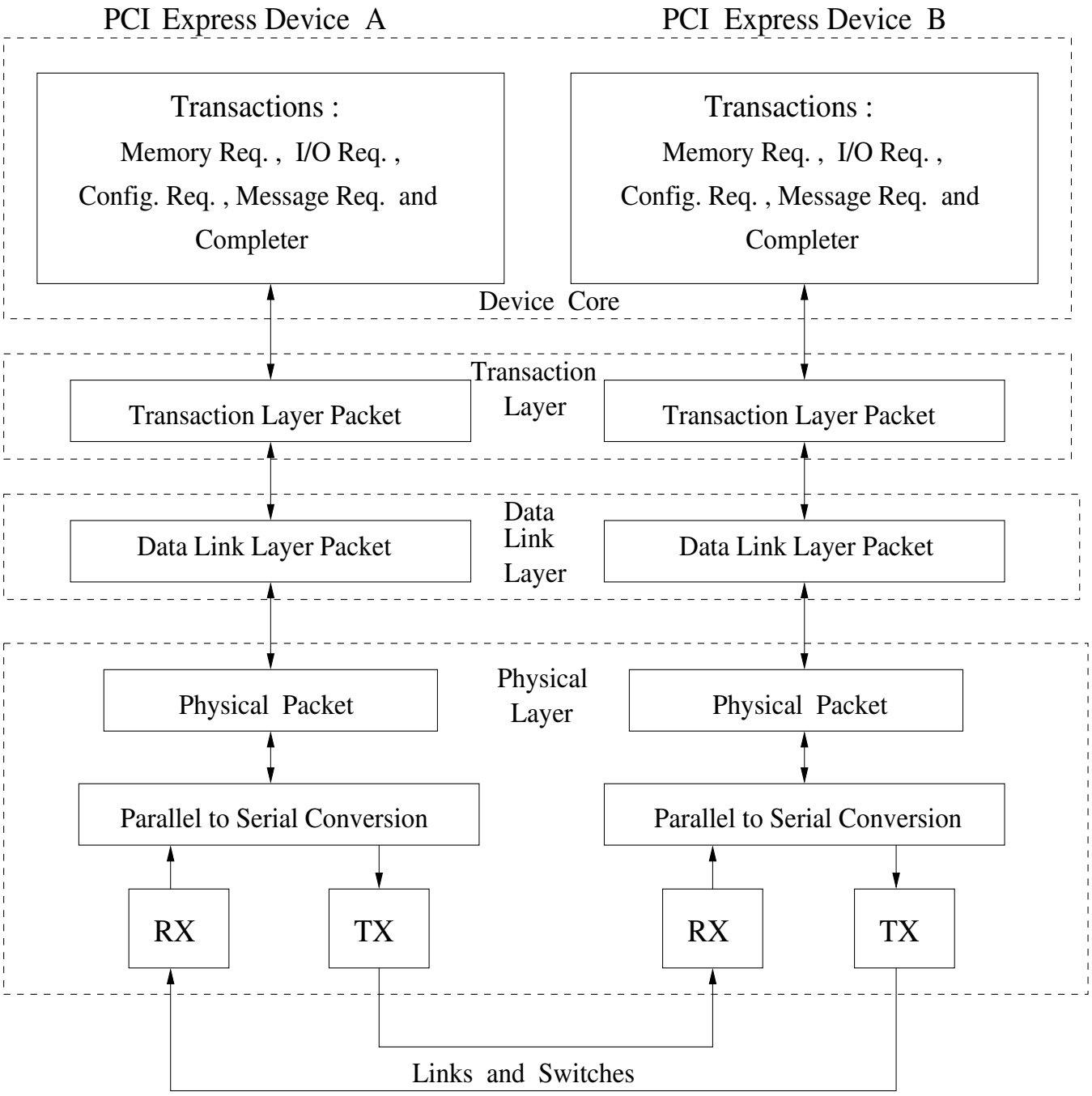


Figure 2.6: Flow of Transaction Layer Packet between two PCI Express Devices

Chapter 3

Transaction Layer Architecture

Transaction Layer's primary responsibility is to create PCI Express request and completion transactions. It has both transmit functions for outgoing transactions and receive functions for incoming transactions. On the transmit side, the Transaction Layer receives request data or completion data from the Device core and then turns that information into an outgoing PCI Express transaction. On the receive side, the Transaction Layer accepts incoming PCI Express transactions from its Data Link Layer. This layer assumes all incoming information is correct because it relies on its Data Link Layer to ensure that incoming information is error-free and received in the proper sequence.

The Transaction Layer uses TLPs (Transaction Layer Packets) to communicate request and completion data with other PCI Express devices. TLPs may address several address space and have a variety of purposes; for example, read versus write, request versus completion and memory versus configuration. Each Transaction Layer Packet has a header assosiated with it to identify the type of transaction. The Transaction Layer of the originating device generates the TLP and the Transaction Layer of the destination device consumes the TLP.

3.1 Transaction Layer Packets

The Transaction Layer Packet (TLP) is the means through which request and completion information is communicated between PCI Express devices. A TLP consists of a header, an optional data payload and an optional TLP digest. The Transaction Layer generates outgoing TLPs based on the information it receives from its device core. The Transaction Layer then passes the TLP on to its Data Link Layer for futher processing. The Transaction Layer also accepts incoming TLPs from its Data Link Layer. The Transaction Layer checks the ECRC (optional) and decodes the header information, and then passes along the appropriate information and data payload to its device core [1].

The TLP always begins with a header. The ***header*** is DWord aligned (a multiple of 4 bytes) but varies in length based on the type of transaction. Depending on the type of packet, Transaction Layer Packets may contain a ***data payload***. If present, the data payload is also DWord aligned for both the first and last DWord of data. To achieve this DWord alignment, DWord byte enable fields within the header indicate whether padding bytes are appended to either the beginning or ending of the data payload. When a data payload is included in a TLP, the first byte of data corresponds to the lowest byte address (i.e. closest to zero) and subsequent bytes of data are in increasing byte address sequence. The TLP may consist of a digest at the end of the packet. The ***TLP digest*** is optional and is used to ensure end-to-end integrity. If used, the digest field contains an ECRC (end-to-end CRC) that ensurses the contents of the TLP are properly conveyed from the source of the transaction to its ultimate destination. The ECRC is a 32-bit value that is generated at the Transaction Layer of the originating device

and checked at the Transaction Layer of the destination device. It incorporates the entire TLP Header and if present, the data payload [1].

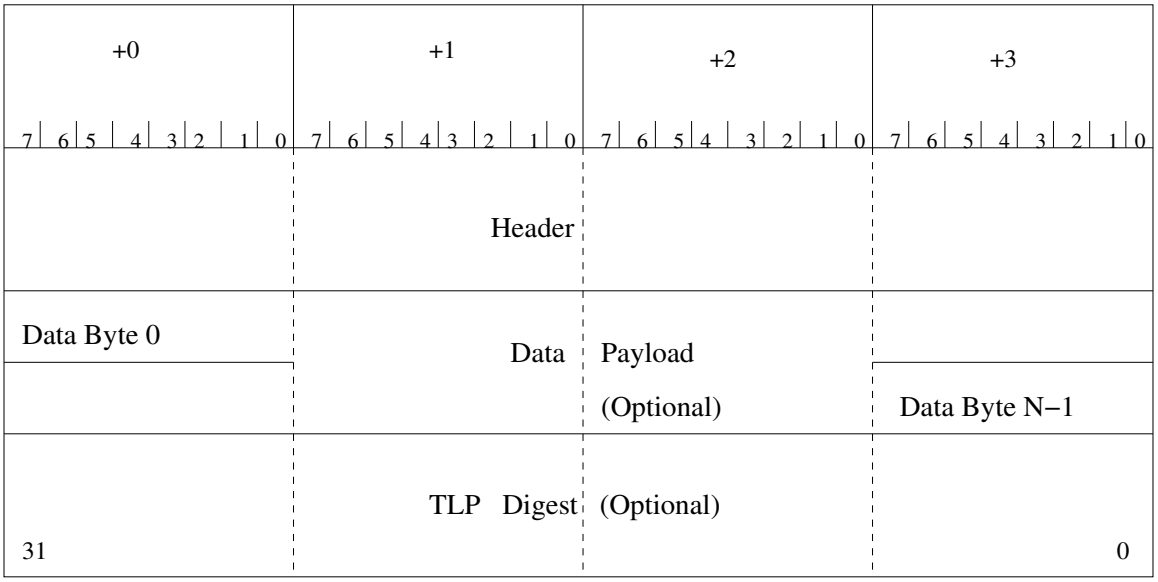


Figure 3.1: Generic TLP Format

3.2 TLP Headers

All TLPs consist of a header that contain the basic identifying information for the transaction. The TLP header may be either three or four DWords in length, depending on the type of transaction. The format of first DWord for all TLP headers is shown in the Figure 3.2.

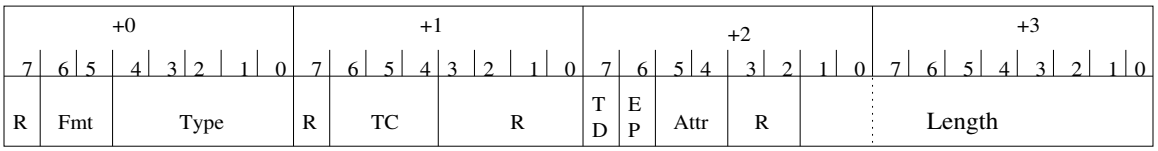


Figure 3.2: Format of First DWord for all TLP Headers

TLP fields marked with an ***R*** indicate a reserved bit or field. Reserved bits are filled with 0's during TLP formation and are ignored by receivers. The ***Fmt*** field indicates the format of the TLP itself. The Fmt field indicates the length of the TLP Header. And the type of the transaction is determined by the combination of the Fmt and ***Type*** fields. The following table shows the assosiated values for the Fmt field [1].

Fmt[1:0] Encoding	TLP Format
00b	3 Dword Header, no data payload
01	4 Dword Header, no data payload
10	3 Dword Header, with data payload
11	4 Dword Header, with data payload

The ***TC*** field indicates the traffic class of the packet. This 3-bit field allows for differentiation of transactions into eight distinct traffic classes. The default traffic class is a value of 000b indicating traffic class TC0. Values between 001b and 111b in TC field indicates traffic classes between TC1 and TC7.

The ***TD*** bit indicates whether a TLP digest is provided at the end of the TLP. A value of 1b in TD field indicates that TLP digest is attached, while a value of 0b indicates that no TLP digest is present. The ***EP*** bit indicates whether a TLP is poisoned. A poisoned TLP is a known bad TLP that is used for the controlled propogation of an error through the system.

The ***Attr*** field contains attributes information for the TLP that allow for traffic handling optimization. The first bit identifies (bit 5 of byte 2) identifies if PCI-X relaxed ordering applies to the TLP. The value of 0b at this bit indicates PCI strongly ordered model and the value of 1b indicates that PCI-X relaxed ordering model applies. The second bit of this field (bit 4 of byte 2) indicates whether cache coherency is required for the transaction.

The ***Length*** field indicates the length of the data payload in DWords. The Length field only identifies the length of the data payload and not of the entire TLP.For example, a value of 0000000001b indicates a data payload that is 1 DWord long. The length of the data payload can vary from 0 to a maximum of 1024 DWords. The overall size of the TLP can be determined from Length field, along with the Fmt (indicating whether the TLP header is three or four DWords) and TD (indicating whether a single DWord digest is attached at the end) fields.

3.2.1 Memory Request Headers

Memory requests are used for normal memory reads, reads to locked memory or for memory writes. Memory requests are also differentiated by the addressing format, as PCI Express supports both 32-bit and 64-bit memory addressing. All memory requests have the common DWord shown in the figure 3.2 as the first DWord of the header. The second DWord for all memory requests contains the same information: the ***Requester ID***, the ***Tag*** and the ***Byte Enable*** fields [1].

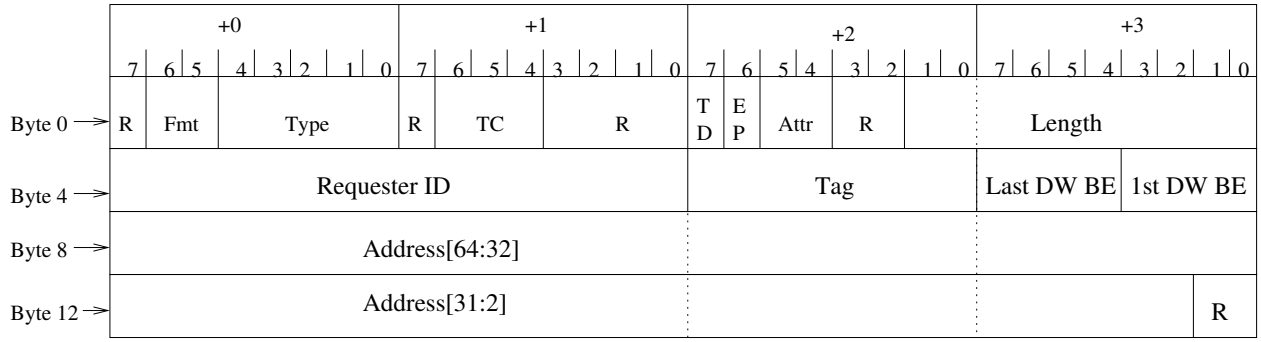


Figure 3.3: 64-bit Address Memory Request Header

The Requester ID field contains the logical bus, device and function number of the requester. This is a 16-bit value that is unique for every PCI Express function within a hierarchy.

The Tag is an 8-bit field that helps to uniquely identify outstanding requests. The requester generates a unique tag value for each of its outstanding requests that requires a completion. Requests that do not require a completion don't have a tag assigned to them. If a completion is required, the requester ID and tag value are copied into the completion header. This allows

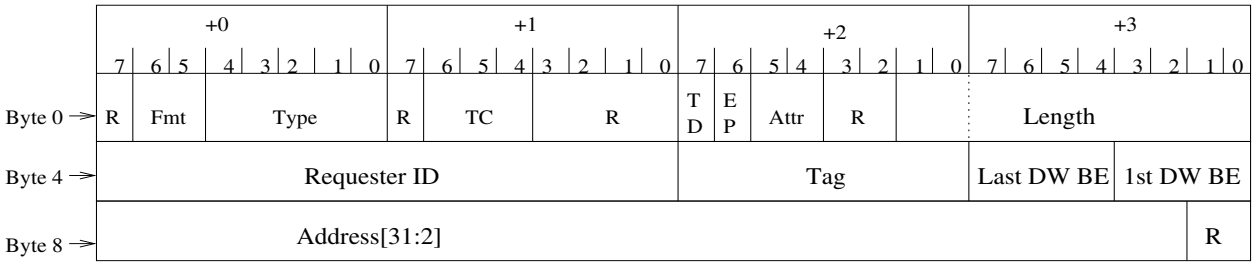


Figure 3.4: 32-bit Address Memory Request Header

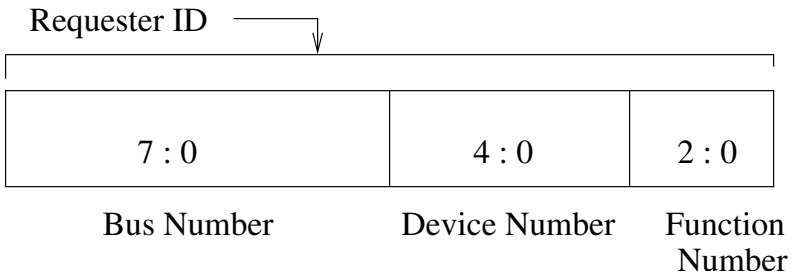


Figure 3.5: Requester ID Format

the system to route that completion packet back to the original requester. The returned tag value identifies which request the completion packet is responding to. These two values form a global identification, called as ***Transaction ID***, that uniquely identifies each request with an accompanying completion. Requests from different devices (of functions within a device) have different requester IDs and multiple requests (that require a completion) from a single device function have different tag values.

The Byte Enable fields contain the byte enables for the first and last DWord referenced by a request TLP. This allows the system to complete data transactions that are not DWord aligned. The First DW BE field contains the byte enables for the first DWord while the Last DW BE contains the byte enables for the last DWord of a request. Each bit within the Byte Enable fields identifies whether its assosiated byte is valid. If the request indicates a length greater than a single DWord, neither the First DE BE field nor the Last DW BE field can be 0000b. Both must specify atleast a single valid byte within their respective DWord. If the request indicates a data length of a single DWord, the Last DW BE field must equal 0000b. If the request is for a single DWord, the First DE BE field can also be 0000b. A memory read request of one DWord with no bytes enabled is referred to as a ***zero length read***.

3.2.2 I/O Request Headers

I/O requests are used for reads or writes to I/O locations. All I/O requests use just the 32-bit address format. All I/O requests have the common DWord shown in the figure 3.2 as the first DWord of the header. The second DWord for all I/O requests utilizes the same format as the second DWord of memory requests, with a Requester ID, Tag and Byte Enables in the exact same locations. Since, all I/O requests are 32-bit addressed, I/O request headers look similar to 32-bit addressed memory request headers.

Since, all I/O requests use 32-bit addressing, all I/O request headers are three DWords in

length. The TC field must always be 000b for all I/O requests. Likewise, the Attr field must have a value of 00b and the Last DW BE field must have a value of 0000b. For I/O requests, the Length field must always have a value of 0000000001b. Figure 3.6 shows the format of the I/O Request Header.

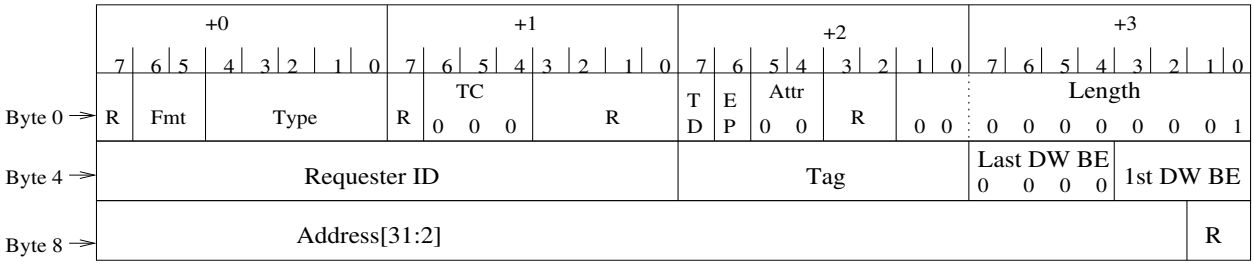


Figure 3.6: I/O Request Header

3.2.3 Configuration Request Headers

Configuration requests are used for reads or writes to Configuration registers of PCI Express devices. Configuration destinations are differentiated based on their bus, device and function numbers, so configuration request packets are routed based on the destination ID and not by address. All configuration requests have the common DWord shown in the figure 3.2 as the first DWord of the header. The second DWord for all configuration requests uses the same format as the second DWord of memory and I/O requests, with the Requester ID, Tag and Byte Enables in the exact same locations. Figure 3.7 shows the format of the Configuration Request Header.

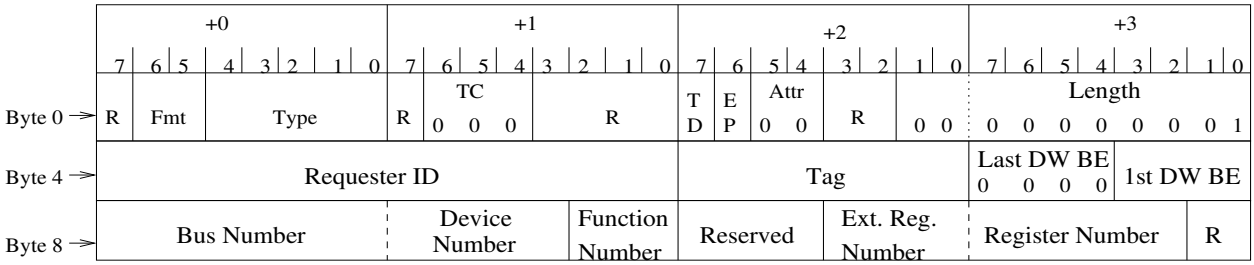


Figure 3.7: Configuration Request Header

All configuration request headers are three DWords in length. The TC field must always be 000b for configuration requests. Likewise, the Attr field must have a value of 00b, the Last DW BE field must have a value of 0000b and the Length field must always have a value of 0000000001b. The ***Bus Number***, ***Device Number*** and ***Function Number*** have the identical format as those used to identify the requester. ***Register Number*** and ***Extended Register Number*** fields are located in the third DWord.

3.2.4 Message Headers

Since PCI Express has no sideband signals, all special events must be transmitted as packets called messages across the PCI Express link. All messages have the common DWord shown in

the figure 3.2 as the first DWord of the header. The second DWord for all messages uses the Transaction ID (Requester ID + Tag) in the same location as memory, I/O and configuration requests. It then adds a message code field to specify the type of message. Figure 3.8 shows the format of the Message Header.

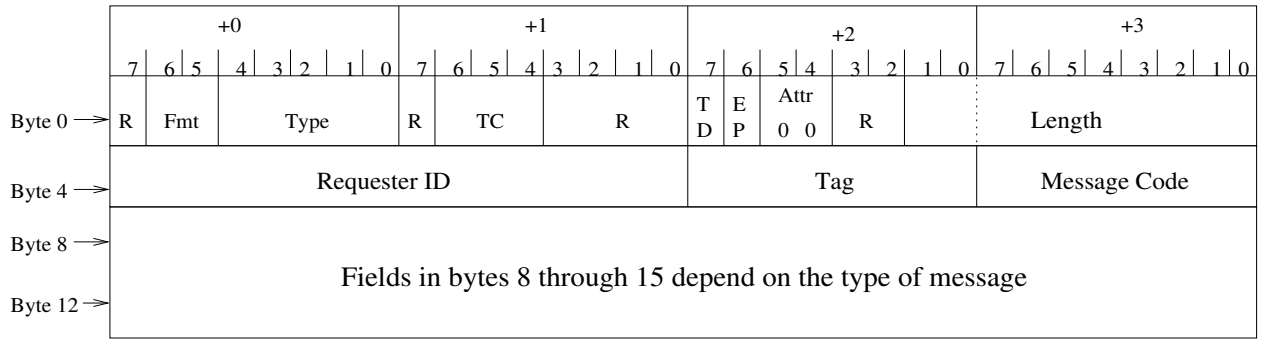


Figure 3.8: Message Header

Interrupt Messages

PCI Express supports interrupts in two different formats: ***INTx emulation*** and ***Message Signaled Interrupt (MSI)*** [1]. MSI interrupt support is required for all PCI Express Devices. For legacy support, devices can emulate the INTx interrupt model through the use of messages. For example, a PCI Express to PCI bridge must translate the INTx signals, the bridge sees on the downstream interface into proper INTx messages on its PCI Express interface. There are eight distinct INTx messages namely: Assert INTA, Assert INTB, Assert INTC, Assert INTD, De-assert INTA, De-assert INTB, De-assert INTC and De-assert INTD. They are differentiated by the Message Code field.

3.3 Completion Packet/Header

Completion packets always contain a completion header and depending on the type of completion, may contain a number of DWords of data as well. Completion Headers are three DWords in length and have the common DWord shown in the figure 3.2 as the first DWord of the header. The second DWord of completion header contains: a ***completer ID***, ***Completion Status***, ***Byte Count Modified (BCM)*** and ***Byte Count***. The third DWord contains the ***Requester ID***, ***Tag value*** and the ***Lower Address Field***. Completion packets are routed by the Requester ID that was supplied with the original request. The Completer ID field is a 16-bit value that is unique for every PCI Express function within the hierarchy. It has the exact same format as the Requester ID. Figure 3.9 shows the format of the Completion Header.

The Completion Status field indicates if the request has been completed successfully. There are four defined Completion Status responses, as shown in the table below.

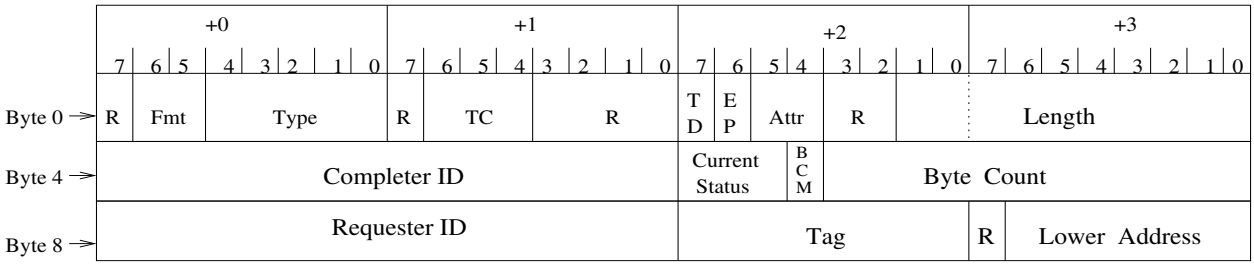


Figure 3.9: Completion Header

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

A single memory read request may result in multiple completion packets. The Byte Count field indicates the remaining number of bytes required to complete a memory read request. It is represented as a binary number with 000000000001b indicating 1 byte and 000000000000b indicating 4096 bytes. This field is used to indicate how many bytes (including the existing completion packet) are still to be returned. If a memory read request is completed with multiple completion packets, the Byte Count field for each successive completiom packet is the value indicated by the previous packet, minus the number of bytes returned with that packet. This field is used to help the requester determine if any of the read completion packets are missing.

The BCM field may be used by PCI-X completers to indicate that the Byte Count field has been modified and is not being used in its normal manner. The Requester ID and Tag fields are copied from the original request packet. The Attr and TC fields must have the same values as the originating request. The completion packet is routed back to the requester based on the Requester ID and the requester then uses the Tag value to identify which request is being completed. The Lower Address Field indicates the byte address for the first enabled byte of data returned with a memory read completion. For any completion other than a memory read, this field must be all 0s.

Chapter 4

Data Link Layer Architecture

The Data Link Layer serves as the ***gatekeeper*** for each individual link within a PCI Express system. It ensures that the data being sent back and forth across the link is correct and received in same order, it was sent. The Data Link Layer makes sure that each packet makes it across the link and makes it across intact.

This layer takes TLPs (Transaction Layer Packets) from the transmit side of the Transaction Layer. It adds a ***sequence number*** to the front of the packet and an ***LCRC*** (link CRC) error checker to the tail and then it forwards this packet on to the Physical Layer [1]. For incoming TLPs, the Data Link Layer accepts the packets from the Physial Layer and checks the sequence number and LCRC to make sure the TLP is correct. If it is correct, the Data Link Layer removes the sequence number and LCRC, then passes the TLP upto the receiver side of the Transaction Layer. If an error is detected, the Data Link Layer does not pass the bad packet onto the Transaction Layer. Instead, the Data Link Layer communicates with its link mate to try and resolve the issue through a rety attempt. The Data Link Layer only passes the TLP to the Transaction Layer, if the packet's sequence number and the LCRC values check out.

4.1 Sequence Number

The Data Link Layer assigns a 12-bit sequence number to each TLP as it is passed from the transmit side of the Transaction Layer [1]. The Data Link Layer applies the sequence number along with a 4-bit reserved field to the front of the TLP. The transmit side of this layer needs to implement two simple counters, one indicating what the next transmit sequence number should be and the one indicating the most recently acknowledged sequence number. When a sequence number is applied to an outgoing TLP, the Data Link Layer refers to the next sequence counter for the appropriate value. And after applying the sequence number, Data Link Layer increments the next sequence counter by one.

The receive side of the Data Link Layer needs to implement a counter for the next receiver sequence number. If the sequence number of the received TLP matches that counter (and the LCRC checks), the Data Link Layer then removes the sequence number, reserved bits and the LCRC. Then, it forwards the incoming TLP onto the receive side of the Transaction Layer. When this occurs, Data Link Layer increments its next receiver sequence counter. If the sequence number does not match the value stored in the receiver's next sequence counter, the Data Link Layer discards that TLP. The Data Link Layer checks to see if the TLP is a duplicate. If it is, it schedules an Ack DLLP to be sent out for that packet. If the TLP is not a duplicate, it schedules a Nak DLLP to report a missing TLP.

The Data Link Layer does not differentiate among types of TLP when assigning the sequence number. Sequence number is used on a link-by-link basis. A TLP has different sequence numbers assosiated with it on the various links, it traverses. The TLP header contains all the

global identifying information but the sequence number only has meaing for a single transmitter and receiver.

4.2 LCRC

The Data Link Layer protects the contents of the TLP by using a 32-bit LCRC value [1]. The Data Link Layer calculates the LCRC value based on the TLP received from the Transaction Layer, the sequence number and the four reserved bits, it has just applied. On the receiver side, the first step that the Data Link Layer takes is to check the LCRC value. If the calculated LCRC value does not equal the received value, the TLP is discarded and Nak DLLP is scheduled for transmission. If the calculated value equals the received value, the Data Link Layer proceeds to check the sequence number. The LCRC protects the contents of a TLP on a link-by-link basis.

4.3 Data Link Layer Packets (DLLPs)

DLLPs support link operations and are strictly assosiated with that given link. DLLPs always originate at the Data Link Layer and are differentiated from TLPs when passed between the Data Link Layer and Physical Layer. A DLLP is always intended for the device on the other side of the link. DLLPs have four major types [1]:

- ***Ack DLLP :***TLP sequence number acknowledgement. These indicate a successful receipt of some number of TLPs.
- ***Nak DLLP :***TLP sequence number negative acknowledgement. These indicate an error condition (for example, a sequence number or LCRC issue, but do not differentiate between the two). Include the last successfully received sequence number and initiates a Data Link Layer retry attempt.
- ***FC DLLPs :***Flow control. The three types of flow control DLLPs are InitFC1, InitFC2 and Update FC. The InitFC1 and InitFC2 are sent during the flow control initialization for each virtual channel. Update FC packets are sent during normal link operation to indicate how much buffer space is available for incoming TLPs.
- ***PM DLLPs :***Power Management DLLPs.

Chapter 5

Physical Layer Architecture

The Physical Layer contains all the necessary digital and analog circuits required to configure and maintain a link. Next generation frequency changes will only require changing the Physical Layer. It eases the transition of upgrading the technology by allowing maximum reuse of the upper layers.

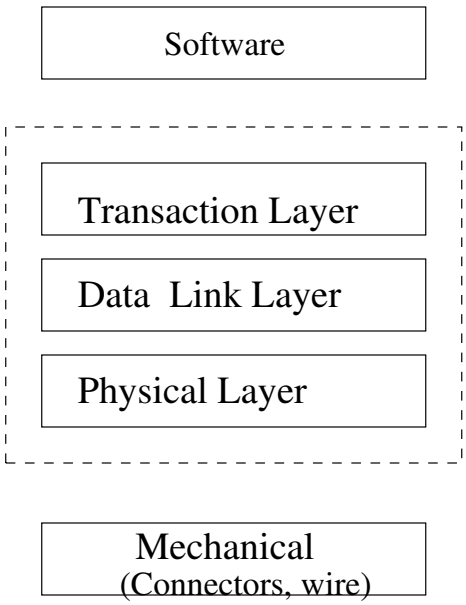


Figure 5.1: Physical Layer Positioning

There are two key sub-blocks that make up the Physical Layer Architecture: a Logical Sub-block and an Electrical Sub-block [1]. Both sub-blocks have dedicated transmit and receive paths that allow dual unidirectional communication between two PCI Express devices.

5.1 Logical Sub-Block

The Logical Sub-block is the key decision maker for the Physical Layer. The logical sub-block has separate transmit and receive paths, referred as ***transmit unit*** and ***receive unit*** [1]. Both are capable of operating indepedently of one another. The primary function of the transmit unit is to prepare packets received from the upper layers for transmission across the link. This process involves three stages: data scrambling, 8-bit/10-bit encoding and packet framing. The receive unit takes the deserialized physical packet received from the wire by the electrical sub-block, removes the framing, decodes it and finally descrambles it.

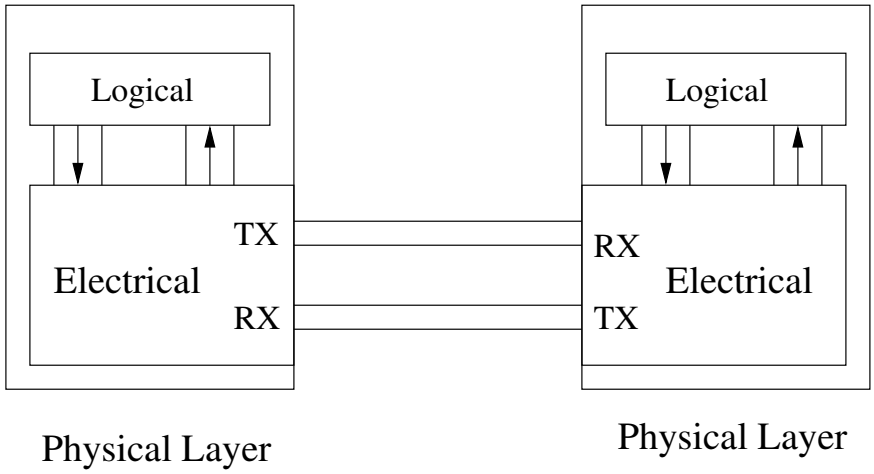


Figure 5.2: Physial Architecture Sub-blocks between two PCI Express Devices

5.1.1 Data Scrambling

Data scrambling is performed to reduce the possiblity of electrical resonance on the link. Most electrical resonance conditions are caused by repeated data patterns at some constant frequency. To avoid this, the PCI Express Base Specification defines a scrambling/descrambling algorithm.

5.1.2 8-Bit/10-Bit Encoding

The 8-bit/10-bit Encoding process invloves converting a byte (8 bits) of data into an encoded 10-bit data symbol. The primary purpose of 8-bit/10-bit encoding is to embed a clock signal into the data stream. By embedding a clock into the data, this encoding scheme renders external clock signals unnecessary. By definition, 8-bit/10-bit encoding allows at most 5 bits of the same polarity to be transmitted before a bit level transition must occur. A byte value represented by bits ABCDEFGH is broken into twp separate bit streams, ABC and DEFGH. Each of these bit streams has a controlled variable appended to it to form a 4-bit and 6-bit stream respectively. Concatenating the 4-bit stream and the 6-bit stream together forms a 10 bit symbol. So, ABC become IABC and DEFGH become JDEFGH, where I and J are control variables.

Encoded data byte and special symbols are generally described by code. For instance, the data byte value 25h is referred to as ***D5.1*** where D stands for Data and 5.1 is related to the byte value 25h. A byte value represented by bits ABCDEFGH is broken into two separate bit streams mainly ABC and DEFGH. The code is formed by taking the decimal equivalent of bits DEFGH, followed by a "." and then the decimal equivalent of bits ABC. So, in case of data byte value 25h (00100101), we get the code as ***D5.1***. Special symbols are coded according to the same process but prefix K is used instead of prefix D.

A secondary benfit of 8-bit/10-bit Encoding is to provide a mechanism for error detection through the concept of Running disparity, which is trying to keep the differnce between the number of transmitted 1s and 0s as close to zero as possible.

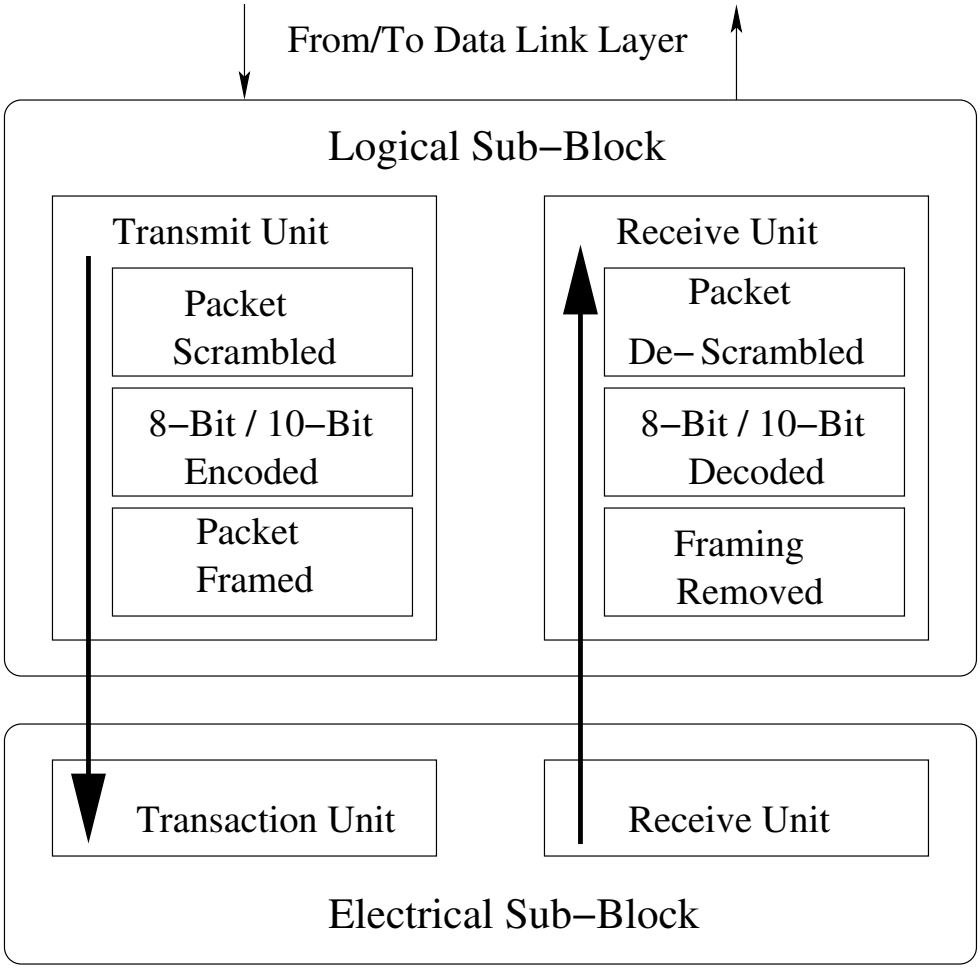


Figure 5.3: Logical Sub-Block Primary Stages

5.1.3 Packet Framing

In order to let the receiving device know where one packet starts and ends, there are identifying 10-bit special symbols that are added and appended to an 8-bit/10-bit Encoded data packet. To end either a TLP or DLLP, the special symbol ***END*** is appended as shown in Figure 5.4..

5.2 Electrical Sub-Block

The electrical sub-block functions as the delivery mechanism for the physical link. This block contains ***transmit*** and ***receive*** buffers that transform the data into/from electrical signals.

5.2.1 Serial/Parallel Conversion

The transmit buffer in the electrical sub-block takes the encoded packetized data from the logical sub-block and converts it into serial format. Once the data is serialized, it is then routed to an assosiated lane for transmission across the link. On the receiver side, the receiver de-serialize the data and feed it back to the logical sub-block for further processing.

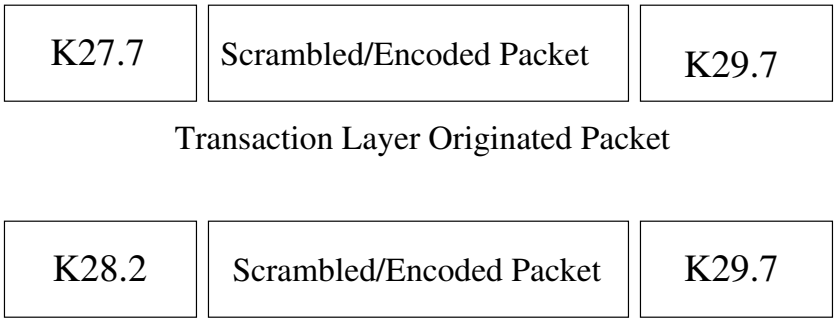


Figure 5.4: Packet Framing Example

Chapter 6

Flow Control

PCI Express enacts flow control (FC) mechanisms to prevent receiver buffer overflow. Flow control is done on a per-link basis, managing the traffic between a device and its link mate. Flow control mechanism do not manage traffic on an end-to-end basis, as shown in Figure 6.1.

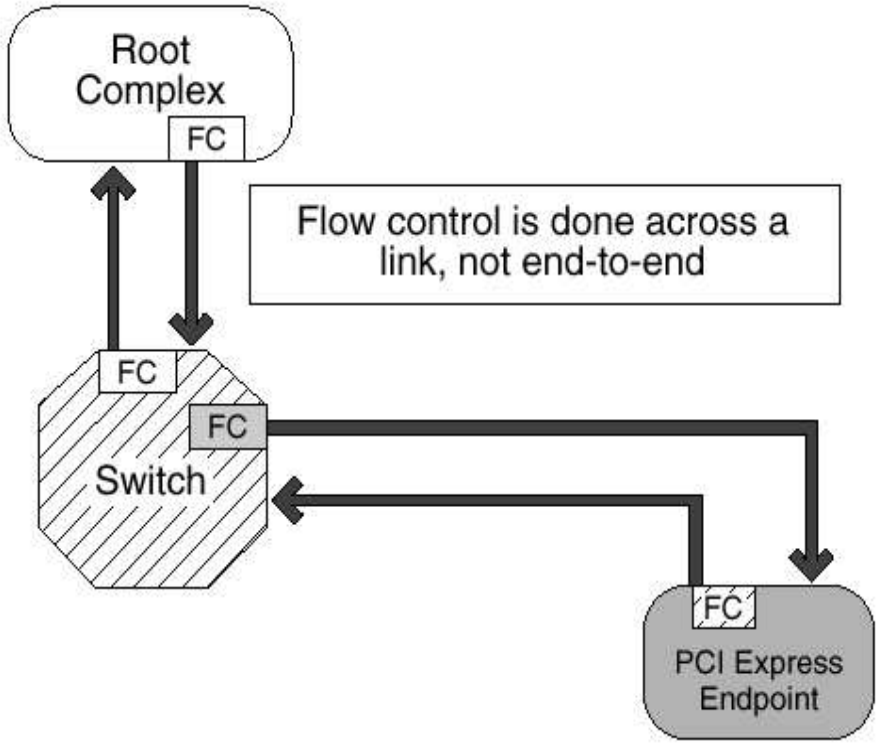


Figure 6.1: Link by Link Flow Control

In the example in Figure 6.1, the root complex issues a request packet destined for the PCI Express endpoint and transmits that packet across the outgoing portion of its link to the switch. The switch then sends that packet across its downstream port to the endpoint. The flow control mechanisms that PCI Express implements, however, are local to each link. The flow control block in the root complex only deals with managing the traffic between the root complex and the switch. The downstream port of the switch and the endpoint then manage the flow control for that packet between the switch and the endpoint. There are no flow control mechanisms in the root complex that track the packet all the way down to the endpoint. Link mates share flow control information to ensure that no device transmits a packet that its link mate is unable to accept. Each device indicates how many flow control credits it has available for use. If the next packet allocated for transmission exceeds the available credits at the receiver, that packet

cannot be transmitted. Within a given link, each virtual channel maintains its own flow control credit pool.

6.1 Virtual Channels and Traffic Classes

One PCI Express lane consists of many ***Virtual Channels***. Each virtual channel can support one or multiple ***Traffic Classes*** [1]. However, a single traffic class cannot be mapped to multiple virtual channels. Each virtual channel has its own set of queues and buffers, control logic and a credit based mecahnism to track how full or empty those buffers are on each side of the link. PCI Express supports upto eight different traffic classes and so eight different virtual channels.

6.2 Flow Control Rules

Consider, a Single lane bridge that must service cars in both directions. To get access to the road, a driver must arbitrate for control of the road with other drivers going both the same direction and the opposite direction. This is a good representation of how conventional PCI and PCI-X flow control works. Additionally, once a car gains access to the road, it needs to determine how fast it can go. If there is a lot of traffic already on the road, the driver has to throttle his or her advancement to keep from colliding with other cars on the road. PCI accomplishes this through signals such as IRDY and TRDY.

Now, consider that the road is changed into a highway with four lanes in both directions. This highway has a carpool lane that allows carpoolers an easier path to travel during rush hour traffic congestion. There are also fast lanes for swifter moving traffic and slow lanes for big trucks and other slow moving traffic. Drivers can use different lanes in either direction to get to a particular destination. Each driver occupies a lane based upon the type of driver he or she is. Carpoolers take the carpool lane while fast drivers and slow drivers occupy the fast and slow lanes respectively. This highway example represents the PCI Express flow control model. Providing additional lanes of traffic increases the total number of cars or bandwidth that can be supported. This is what is accomplished by adding additional lanes to a PCI Express link. Prioritizing who gets to use the available bandwidth (especially during high traffic times) is what virtual channels and traffic classes add to the picture.

6.3 Flow Control Credits

PCI Express uses a Flow Control Credit Model. Data Link Layer Packets (DLLPs) are exchanged between link mates indicating how much free space is available for various types of traffic. This information is exchanged at initialization, and then updated throughout the active time of the link. The exchange of this information allows the transmitter to know how much traffic it can allow on to the link, and when the transmitter needs to throttle that traffic to avoid an overflow condition at the receiver. Flow control differentiates between various types of TLPs (Transaction Layer Packets) and allocates separate credit pools for each type.

TLPs are divided up into the following types for flow control purposes : ***posted request header (PH)***, ***posted request data (PD)***, ***non-posted request header (NPH)***, ***non-posted request data (NPD)***, ***completion header (CplH)*** and ***completion data (CplD)*** [1]. Posted request credits (Px) apply to message and memory write requests. Non-posted

request credits (NPx) apply to IO and configuration write, as well as all read requests. Completion credits (Cplx) apply to the completions associated with a corresponding request. For the various data credits, the corresponding unit of credit is equal to 16 bytes of data (that is to say that 1 CplD unit equals 16 bytes of completion data). For the various header credits, the corresponding unit of credit is the maximum-size header plus TLP digest. Table 6.2, identifies the credits associated with the various types of traffic.

TLP	Credits Consumed
Memory, I/O, configuration read request	1 NPH
Memory write request	1 PH + n PD
I/O, configuration write request	1 NPH + 1 NPD (note: size of data written for these TLPs is never more than one aligned DWord)
Message request without data	1 PH
Message request with data	1 PH + n PD
Memory read completion	1 CplH + n CPLD
I/O, configuration read completions	1 CplH + 1 CPLD
I/O, configuration write completions	1 CplH

Figure 6.2: TLP Flow Control Credits

6.4 Flow Control at the Transmitter

For each credit type, [1] there are two quantities that the transmit side of the Transaction Layer must check to properly implement flow control:

- ***Credits-Consumed***
- ***Credits-Limit***

Credits-Consumed, which is initially set to 0 after initialization, tracks the total number of flow control credits that have been consumed by TLPs. Credit-Limit indicates the most recent number of flow control units legally advertised by a link-mate. This value is first set during flow control initialization and may be updated via UpdateFC packets.

Prior to transmitting any TLP, the Transaction Layer must first determine if there are sufficient outstanding credits for that TLP. If the transmitter does not have enough credits, it must block the transmission of that TLP. This may stall other TLPs that use that same virtual

channel. If there are not enough flow control credits for transmission of a given transaction, transactions that use other traffic classes/virtual channels are not impacted.

It should be noted that return of flow control credits does not necessarily mean that the TLP has reached its final destination or that the associated request/completion has been processed. It simply means that the buffer or queue space allocated to that TLP at the receiver has been cleared. In Figure 6.1, the upstream port of the switch may send an UpdateFC that indicates it has freed up the buffer space from a given TLP that is destined for the endpoint. The root complex should not imply that this has any meaning other than that the TLP has been cleared from the upstream receive buffers of the switch. That TLP may be progressing through the core logic of the switch, may be in the outgoing queue on the downstream port, or may be already received down at the endpoint.

6.5 Flow Control at the Receiver

For each credit type, there is one quantity that the receiver side of the Transaction Layer must check to properly implement flow control,

Credits-Allocated. The Transaction Layer may optionally implement a

Credits-Received register/counter [1]. Credits-Allocated is initially set according to the buffer size and allocation policies of the receiver. This value is included in the InitFC and Update FC packets that are sent across to a link mate. This value is incremented as the receiver side of the Transaction Layer frees up additional buffer space by processing received TLPs. Credits-Received is an optional error checking register or counter. Here, the Transaction Layer simply counts up all the credits (of a given type) that have been successfully received. If the receiver implements this option, it can check for receiver overflow errors (TLPs that exceeded the Credits-Allocated limit).

For finite NPH, NPD, PH, and CplH credit types, an UpdateFC packet must be scheduled for transmission if all advertised flow control units for a particular type are consumed by received TLPs. Additionally, for all finite credit types, UpdateFC packets are required if one or more units of that type are made available by processing received TLPs. UpdateFC packets may be scheduled for transmission more often than required. For a given implementation, it is possible that queues need not be physically implemented for all credit types on all virtual channels. For unimplemented queues, the receiver can advertise infinite flow control during initialization to eliminate the appearance of tracking flow control credits for that type.

6.6 An Example of Flow Control Credits

At initialization, Device B indicates that it has 04h PH credits and 040h PD credits. Device A puts those in its PH and PD Credit-Limit counters/registers. After initialization, Device A has set its PH and PD Credits-Consumed counters/registers to zero. Device A then sends out two P requests (with sequence numbers 1 and 2) that each utilizes a single PH unit and 10h PD units. It therefore updates its PH and PD Credits-Consumed counters/registers to 02h and 20h, respectively.

Device A now wants to send out another P request (sequence number 3) that uses a single PH unit and 30h PD units. In this example, however, the Transaction Layer of Device A must stop that TLP and not transmit it just yet. For while it has the necessary PH credits for this transaction, it does not have the proper number of PD credits left. Device B originally

advertised support for 040h PD units and Device A has already sent out packets that consumed 020h of that. That leaves only 020h credits available, not enough to cover the 030h that TLP with sequence number 3 requires. Device A must stop this TLP. Once Device B issues an UpdateFC-P packet that indicates that one or both of the outstanding TLPs (sequence numbers 1 and/or 2) has been cleared from its queues, Device A can release TLP with sequence number 3 and transmit it as appropriate.

Chapter 7

Proposed Design Outline

The Second Stage of this Project includes the Implementation of FPGA - based PCI / PCI Express Bridge. The Implementation part includes VHDL Coding for the Design and targeting the design on FPGA Kit. I am giving the Proposed Design Outline for the Implementation in Figure 7.1.

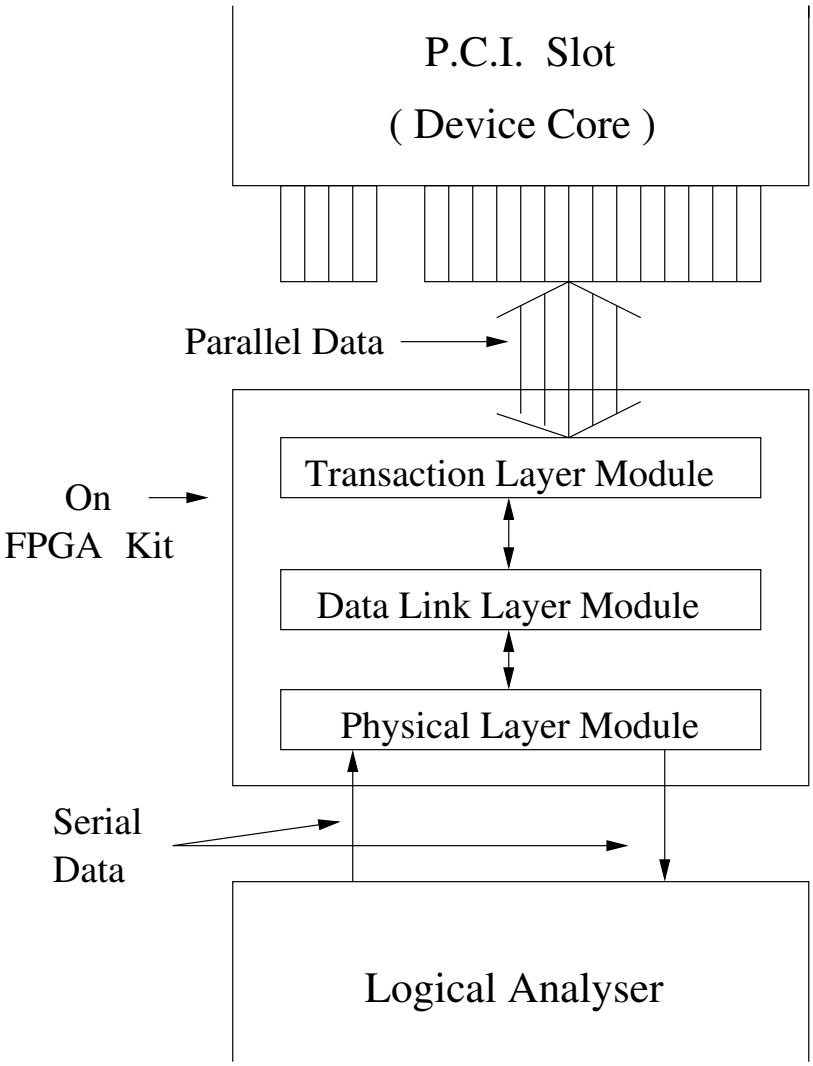


Figure 7.1: Proposed Design Outline for the Implementation

The FPGA Kit is connected to PCI Slot on one side and Logical Analyser on other side. The VHDL Code on the FPGA Kit consists of 3 modules. These 3 modules are :

- Transaction Layer Module
- Data Link Layer Module
- Physical Layer Module

Transaction Layer Module is responsible to act as Transaction Layer. It gets Parallel Data from PCI Slot (Device Core) and identifies the type of Transaction and other information from other signals from the PCI Slot. It generates the Header accordingly and generates the ECRC if required. It appends the Header to the front of the Data and ECRC at the end of the Data. It then passes this packet to Data Link Layer Module. ***Data Link Layer Module*** acts as Data Link Layer. It generates the proper Sequence Number and appends it to the front of the packet, it has got from Transaction Layer Module. Then, it generates the LCRC for this packet and appends this LCRC at the end of the packet. It then passes this packet to the ***Physical Layer Module***, which acts as Physical Layer. It is responsible for 8 Bit/10 Bit Encoding and Packet framing. Then, it serializes the packet and transmit the serial data to the Logical Analyser.

Physical Layer Module also receives the serial data from Logical Analyser. It de-serializes the data, then it removes framing and then do 8 Bit/10 Bit Decoding. It passes this packet to the Data Link Layer Module. This module checks for LCRC and Sequence Number to ensure that the packet is correct. If it is not correct, it does not pass the bad packet to the Transaction Layer Module and do further processing. If it is correct, it removes sequence number and LCRC, and sends the remaining packet to the Transaction Layer Module. This module checks for ECRC. Then, it removes ECRC and decodes the Header. It then generates the Parallel data and accordingly transfers this data and other information to the PCI Slot (Device Core).

Chapter 8

Conclusion and Future Work

The adoption of P.C.I. Express will serve as the Point of Inflection that will forever change the course of events for the computing and communication sectors. The P.C.I. Express Initiative is on the track and rapidly gaining momentum in the Industry. About 90 percent of the Computers are still based on P.C.I. So, there is an urgent need to either replace the entire platform to P.C.I. Express or to develop a Bridge which can convert P.C.I. to P.C.I. Express and vice versa. So, my future work is to Design and Implement an FPGA-based P.C.I / P.C.I. Express Bridge.

Bibliography

[1] Justin Schade Adam Wilen and Ron Thornburg. *Introduction to P.C.I. Express, A Hardware and Software Developer's Guide.* INTEL PRESS, 2003.

[2] Tom Shanley and Don Anderson. *P.C.I. System Architecture.* MINDSHARE, INC., 1999.

[3] Edward Solari and Brad Congdon. *The Complete P.C.I. Express Reference, Design Insights for Hardware and Software Developers.* INTEL PRESS, 2003.