DESIGN AND IMPLEMENTATION OF PCI-E CORE WITH MIX LANGUAGES

By

Abd-Elrahman Badr Diaa El-Din

Abd-Elrahman Khaled Abd-Elrahman

Marwa Sayed Ahmed Aboouf

Mohamed Yasser Abd-Elmotaleb

Under the Supervision of Associate Prof. Hassan Mostafa

A Graduation Project Report Submitted to

The Faculty of Engineering at Cairo University in

Partial Fulfillment of the Requirements for the

Degree of Bachelor of Science in

Electronics and Communications Engineering Faculty of Engineering,

Cairo University Giza, Egypt

Department of Electronics and
Electrical Communications Engineering
Faculty of Engineering - Cairo University

Abstract

According to Moore's Law, processing speeds have doubled every 18 months since the invention of the integrated circuit. Also, the huge technological improvement leads us to use protocols of transferring data with high speed. PCI (Peripheral Component Interconnect) Express bus was introduced by Intel in 2004. That protocol helps us to reach high throughput of data, lower I/O pins and more detailed error detection and reporting mechanism (Advanced Error Reporting, AER). PCI-E represents the currently fastest and most expensive solution to connect the peripheral devices with general purpose CPU. It provides highest bandwidth connection in the PC platform.

Keywords: PCI, PCI-E, Moore's Law.

Acknowledgements

We are using this opportunity to express our gratitude to everyone who supported us throughout the graduation project. We are thankful for their aspiring guidance and friendly advice.

First, we want to thank our major advisor Dr. Hassan Mustafa for his encouragement through the whole year, his caring about following up each stage in the project and his suggestions to solve some problems we faced during the project work.

We want to thank team PCI Mentor Graphics, Eng. Mahmoud El-Tahawy, Team Leader, and Eng. Ahmed Khedr, and Eng. Ahmed El-Zeiny, and Eng. Adham Rageh, for providing their time, experience to help us overcome some obstacles we faced during some stages especially when dealing with new concepts and tools.

Finally, we want to thank our families for their support, tolerance and love during this year especially during the hard times they were always there having faith in what we do. We are grateful to our families, colleagues and friends for always motivating us, without them we wouldn't have come so far.

Table of Contents

List of figures

List of Acronyms

ACK	Acknowledgement
AER	Advanced Error Reporting
AGP	Accelerated Graphics Port
Attr	Attributes
BE	Byte Enable
BIST	Built-In-Self-Test
CA	Completion Abort
CfgRd	Configuration Read
CfgWr	Configuration Write
Cpl	Completion
CplD	Completion with Data
CRC	Cyclic Redundancy Check
CRS	Configuration Retry Request
CXL	Compute Express Link
DLLP	Data Link Layer Packet
DMA	Direct Memory Access
DW	Double Word
EISA	Extended Industry Standard Architecture
EP	End Point
FC	Flow Control
FCC	Flow Control Credit
IDE	Integrated Drive Electronics
INTx	Interrupt
ISA	Industry Standard Architecture

MCA	Micro Channel Architecture
MemRd	Memory Read
MemWr	Memory Write
MSI	Message Signaled Interrupt
Msg	Message
MsgD	Message with Data
NRS	Next Receive Sequence
NTS	Next Transmit Sequence
OS	Operating System
PCI	Peripheral Component Interconnect
PCI-E	Peripheral Component Interconnect Express
PCI-SIG	Peripheral Component Interconnect Special Interested Group
PCI-X	Peripheral Component Interconnect eXtended
PLP	Physical Layer Packet
QoS	Quality of Service
RC	Root Complex
SC	Successful Completion
SSD	Solid State Drive
TC	Traffic Class
TD	TLP Digest
TLP	Transaction Layer Packet
UR	Unsupported Request
VC	Virtual Channel
VESA	Video Electronics Standards Association
VGA	Video Graphics

Chapter 1

Introduction

In this thesis, we are going to propose the design and topology of PCI-E (Peripheral Component Interconnect Express) Core implemented in mix language with DPI protocol.

1-1 Motivation

PCI-E is a high-speed serial computer expansion bus standard, designed to replace the older PCI, PCI-X and AGP bus standards. It is the common motherboard interface for personal computers' graphics cards, hard drives, SSDs, Wi-Fi and Ethernet hardware connections.

1-2 Architecture

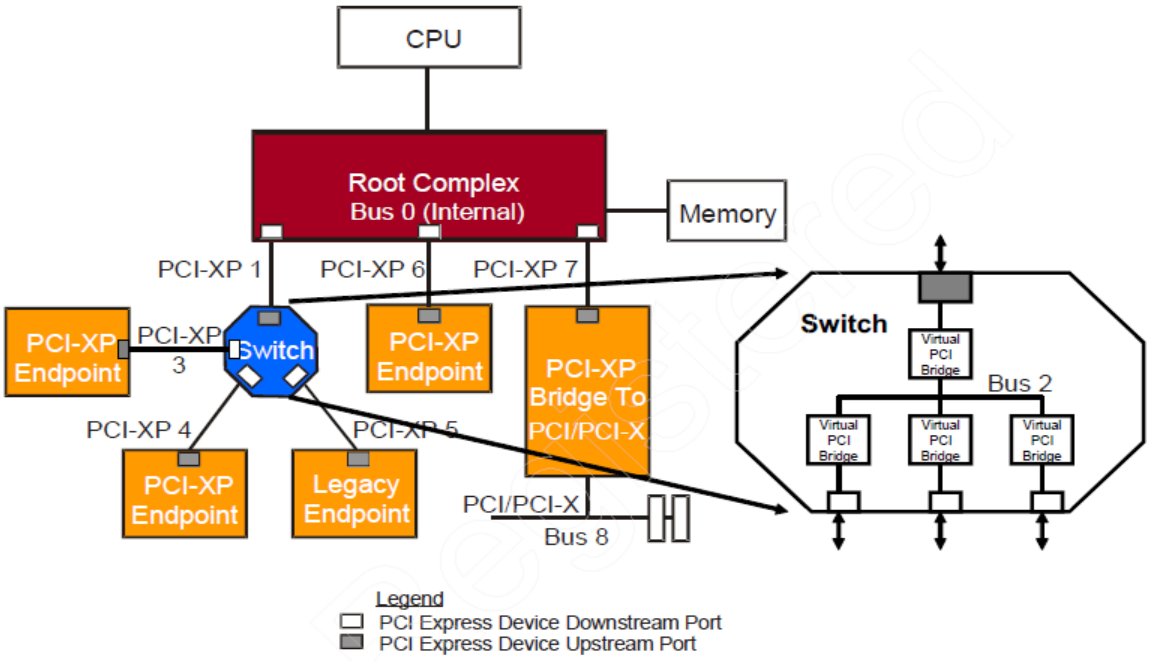


Figure 1: Topology of PCI-E

The Root Complex denotes the device that connects the CPU and memory sub-system to the PCI Express fabric. It may support one or more PCI Express ports. The root complex in this example supports 3 ports. Each port is connected to an endpoint device or a switch which forms a sub-hierarchy. The root complex generates transaction requests on the behalf of the CPU. It is capable of initiating configuration transaction requests on the behalf of the CPU. It generates both memory and IO requests as well as generates locked transaction requests on the behalf of the CPU. The root complex as a completer does not respond to locked requests. Root complex transmits packets out of its ports and receives packets on its ports which it forwards to memory. A multi-port root complex may also route packets from one port to another port but is NOT required by the specification to do so.

Root complex implements central resources such as: hot plug controller, power management controller, interrupt controller, error detection and reporting logic. The root complex initializes with a bus number, device number and function number which are used to form a requester ID or completer ID. The root complex bus, device and function numbers initialize to all 0s.

PCI Express devices communicate via a logical connection called an interconnect or **link**. A link is a point-to-point communication channel between two PCI Express ports allowing both of them to send and receive ordinary PCI requests (configuration, I/O or memory read/write) and interrupts (INTx, MSI or MSI-X). At the physical level, a link is composed of one or more **lanes** (from x1 to x32 lanes).

A **lane** is composed of two differential signaling pairs, with one pair for receiving data and the other for transmitting. Thus, each lane is composed of four wires or signal traces.

Rather than bus cycles we are familiar with from PCI and PCI-X architectures, PCI Express encodes transactions using a packet-based protocol. Packets are transmitted and received serially and byte striped across the available Lanes of the Link. The more Lanes implemented on a Link the faster a packet is transmit-ted and the greater the bandwidth of the Link. The packets are used to support the split transaction protocol for non-posted transactions. Various types of packets such as memory read and write requests, IO read and write requests, configuration read and write requests, message requests and completions are defined.

1-3 Applications

- It can be used as peripheral device interconnect, chip-to-chip interface and as a bridge to many other protocol standards.
- Expansion Card interface.
- Sound Card.
- Network Cards (wired and wireless).
- GPUs.
- Interface in High-Performance Video Systems.
- Storage products like an SSD, to this high bandwidth interface allows for much faster reading from, and writing to.

1-4 Generations

Table 1: Generations of PCI-E

Generation	introduced	BW / Frequency	Line Code	Transfer Rate	Throughput				
					x1	x2	x4	x8	x16
One	2003	8 GB/s 2.5 GHz	8b/10b	2.5 GT/s	250 MB/s	0.5 GB/s	1 GB/s	2 GB/s	4 GB/s
Two	2007	10 GB/s 5 GHz	8b/10b	5 GT/s	500 MB/s	1 GB/s	2 GB/s	4 GB/s	8 GB/s
Three	2010	32 GB/s 8 GHz	128b/130b	8 GT/s	984.6 MB/s	1.969 GB/s	3.938 GB/s	7.88 GB/s	15.75 GB/s
Four	2017	64 GB/s 16 GHz	128b/130b	16 GT/s	1969 MB/s	3.938 GB/s	7.88 GB/s	15.75 GB/s	31.51 GB/s
Five	2019	128 GB/s 32 GHz	128b/130b	32 GT/s	3938 MB/s	7.88 GB/s	15.75 GB/s	31.51 GB/s	63.02 GB/s
Six (Planned)	2021	256 GB/s 32 GHz	128b/130b (PAM4)	64 GT/s	7877 MB/s	15.75 GB/s	31.51 GB/s	63.02 GB/s	126.03 GB/s

PCI-E has five generations and Gen6 will be released in 2021.

As shown in the above table, Throughput of generations doubles because the frequency doubles also, despite PCI-E Gen5 and Gen6 which have the same frequency. That's because Gen6 switches to PAM4 (Pulse Amplitude Modulation) so that 2 bits are transferred per transfer.

1-5 PCI Express Device Layers

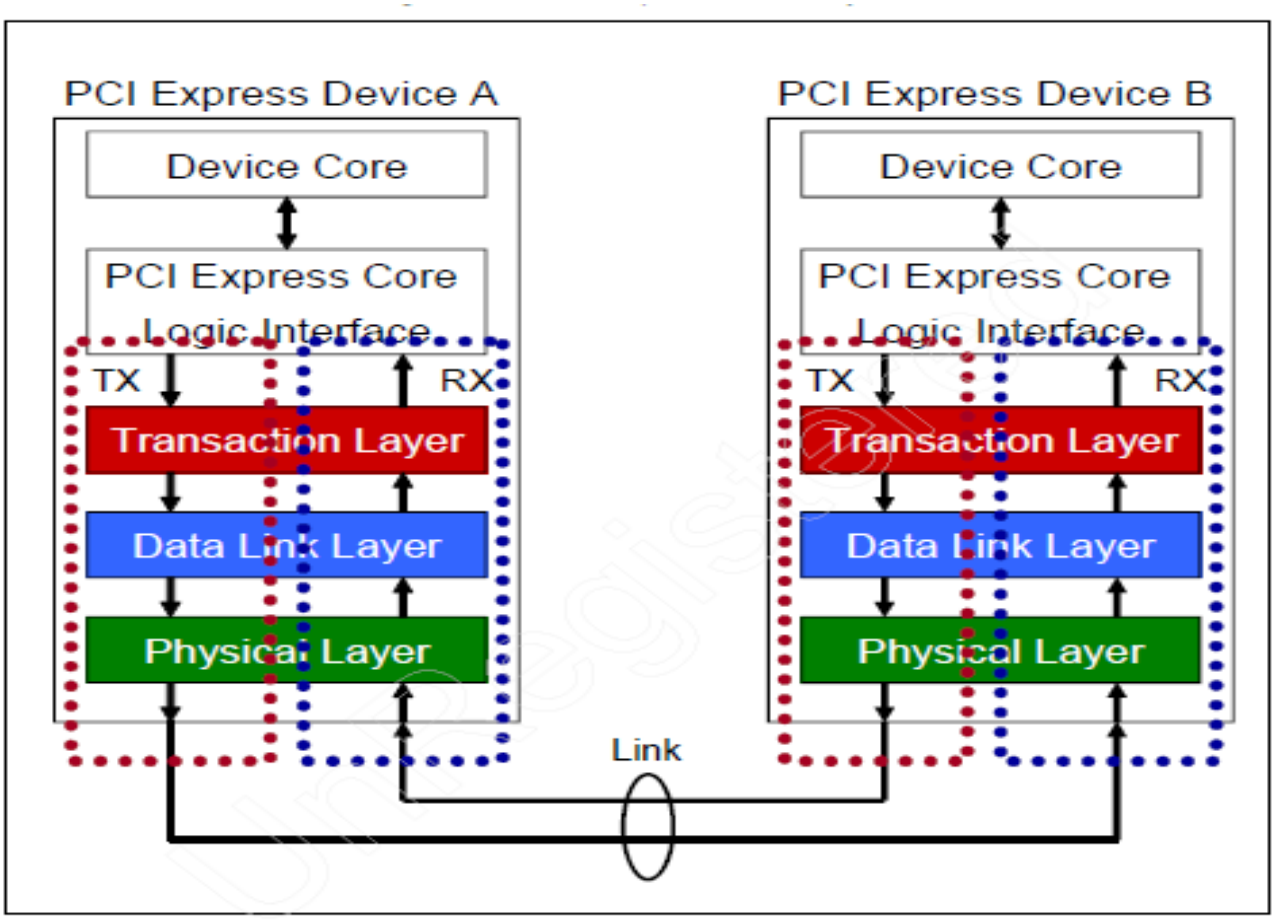


Figure 2: PCI Express Device Layers

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. 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.

Chapter 2

History of Buses

2-1 ISA (Industry Standard Architecture) Bus

2-1-1 Introduction

ISA is a standard bus architecture that was associated with the IBM AT motherboard. In 1981, the first IBM personal computers (PCs) introduced included the 8-bit subset of the ISA bus, and the PC AT, which IBM launched in 1984, was the first full 16-bit implementation of the ISA bus. It allowed 16 bits at a time to flow between the motherboard circuitry and an expansion slot card and its associated device(s).

ISA was one of the first expansion buses for PCs. Providing the hardware interface for connecting peripheral devices in PCs, ISA accepted cards for sound, display, hard drives and other devices. ISA allowed for additional expansion cards to be attached to a computer's motherboard and was capable of direct memory access (DMA), with multiple expansion cards on a memory channel and separate interrupt request (IRQ) assignment for each card. The development and use of ISA led to several later technologies.

2-1-2 Specifications

- 8-bit or 16-bit Data width
- Maximum Clock Frequency 8 MHz synchronous with the CPU clock
- Parallel internal bus with a separate (not multiplexed) address and data signals
- 8 Mbytes/Sec Speed
- 8-bit ISA Bandwidth is 8 MHz && 16-bit ISA Bandwidth is 16 MHz
- Used devices → Sound cards, Modems, Network cards, display and hard drives ....

2-1-3 Future Replacement

In the early 1990s, Intel developed PCI, which combined the characteristics of ISA and VL-Bus. PCI provided direct access to system memory for connected devices while employing a bridge to connect to the front side bus and, thus, to the CPU. As a result, PCI's performance exceeded the VL-Bus while eliminating the potential for interference with the CPU.

By the mid-1990s, new motherboards were manufactured with fewer ISA slots, and PCI became standard for connecting computers and their peripherals. For several years, motherboards had a combination of 8-bit and 16-bit ISA slots. As PCI became popular,

motherboards included only 16-bit ISA and PCI. Yet, by the early 2000s, the PCI interface replaced ISA.

2-2 EISA (Extended Industry Standard Architecture) Bus

2-2-1 Introduction

Extended Industry Standard Architecture (EISA) also known as Extended ISA is a bus architecture that extends the Industry Standard Architecture (ISA) from 16 bits to 32 bits. EISA was introduced in 1988 by the Gang of Nine - a group of PC manufacturers.

2-2-2 Specifications

- 32-bit Data width
- Clock Frequency 8.33 MHz synchronous with the CPU clock
- Parallel internal bus with a separate (not multiplexed) address and data signals
- 33 Mbytes/Sec Speed
- Bandwidth is 33 MHz
- Used devices → Servers, network interface cards (NIC) or small computer system interfaces (SCSI)…

2-2-3 Future Replacement

Eventually, PCs required faster buses for higher performance. Faster expansion cards, like Local Bus or Video Electronics Standards Association (VESA), were introduced, and there was no longer an EISA card market. EISA bus is no longer used and replaced with PCI bus as EISA was expensive so it was used in servers.

2-3 MCA (Micro Channel Architecture) Bus

2-3-1 Introduction

MCA was introduced by IBM in 1987 as a competitor to the ISA bus it has 2 versions ,version for 16 bit and version for 32 bit.

There are two types, 16- or 32-bit parallel Internal computer bus however, as both the 32-bit and 16-bit versions initially had a number of additional optional connectors for memory cards which resulted in a huge number of physically incompatible cards for bus attached memory. In time, memory moved to the CPU's local bus, thereby eliminating the problem. On the upside, signal quality was greatly improved as Micro Channel added ground and power pins and arranged the pins to minimize interference; a ground or a supply was thereby located within 3 pins of every signal.

2-3-2 Specifications

- bus speed of MCA is 10 MHZ.
- Data width of MCA is 32 or 16 bits.
- Band width of MCA is 40 MB/s.

2-3-3 Future Replacement

It was used on PS/2 and other computers until the mid-1990s. Its name is commonly abbreviated as "MCA", although not by IBM. In IBM products, it superseded the ISA bus and was itself subsequently superseded by the PCI bus architecture.

2-4 VESA (Video Electronics Standards Association) Bus

2-4-1 Introduction

VLB or VL bus is short for VESA (Video Electronics Standards Association) local bus. Introduced by VESA in 1992, the VLB is a 32-bit computer bus that had direct access to the system memory at the speed of the processor. The 486 CPU (33/40 MHz) employed the VLB. VLB 2.0 was released in 1994, with a 64-bit bus, and a bus speed of 50 MHz, Unfortunately, the VLB relied heavily on the 486 processor. When the Pentium processor was introduced, manufacturers began switching to PCI Bus Type.

2-4-2 Specifications

- For bus speed 25 MHz bandwidth is 100 MB/s
- For bus speed 33 MHz bandwidth is 133 MB/s
- For bus speed 40 MHz bandwidth is 160 MB/s
- For bus speed 50 MHz bandwidth is 200 MB/s (out of specification)
- Data width as I said before 32-bit for version 1 and 64-bit for version 2

2-4-3 Future Replacement

Despite these problems, the VESA Local Bus became very commonplace on later 486 motherboards, with a majority of later (post-1992) 486-based systems featuring a VESA Local Bus video card. VLB importantly offered a less costly high-speed interface for mainstream systems, as only by 1994 was PCI commonly available outside of the server market through the Pentium and Intel's chipsets. PCI finally displaced the VESA Local Bus (and also EISA) in the last years of the 486 market, with the last generation of 80486 motherboards featuring PCI slots instead of VLB-capable ISA slots. However, some manufacturers did develop and offer "VIP" (VESA/ISA/PCI) motherboards with all three slot types.

2-5 PCI (Peripheral Component Interconnect) Bus

2-5-1 Introduction

PCI is parallel internal bus. It was created by intel in 1992, PCI local bus is the general standard for a PC expansion bus, having replaced the Video Electronics Standards Association (VESA) local bus and the Industry Standard Architecture (ISA) bus.

The first version of PCI found in retail desktop computers was a 32-bit bus using a 33 MHz bus clock and 5 V signaling, although the PCI 1.0 standard provided for a 64-bit variant as well. These have one locating notch in the card. Version 2.0 of the PCI standard introduced 3.3 V slots, physically distinguished by a flipped physical connector to prevent accidental insertion of 5 V cards. Universal cards, which can operate on either voltage, have two notches. Version 2.1 of the PCI standard introduced optional 66 MHz operation. A server-oriented variant of PCI, called PCI-X (PCI Extended) operated at frequencies up to 133 MHz for PCI-X 1.0 and up to 533 MHz for PCI-X 2.0. An internal connector for laptop cards, called Mini PCI, was introduced in version 2.2 of the PCI specification. The PCI bus was also adopted for an external laptop connector standard –the Card Bus. The first PCI specification was developed by Intel, but subsequent development of the standard became the responsibility of the PCI Special Interest Group (PCI-SIG).

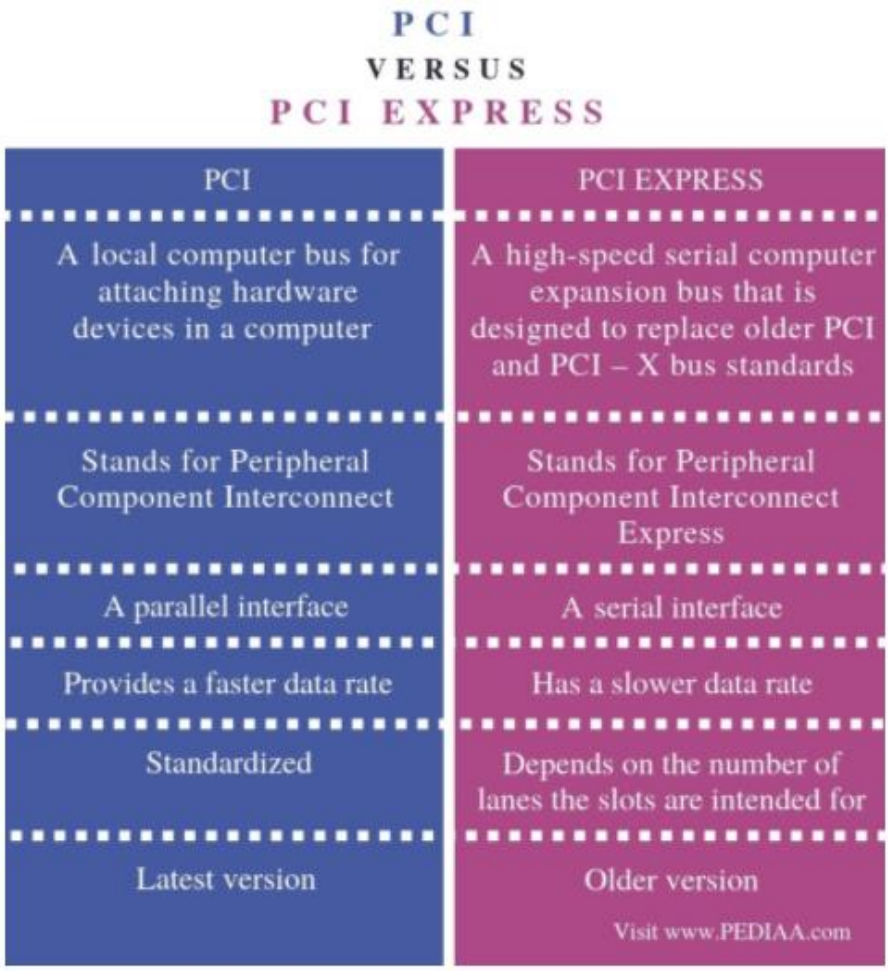
2-5-2 Future Replacement

PCI was superseded by PCI express in 2004 and PCIe becomes common.

2-5-3 Difference between PCI and PCI-E

This table shows some differences between PCI and PCI-E

Table 2: Difference between PCI and PCI-E



- PCIe is much faster compared to PCI.
- PCIe uses a serial interface while PCI uses a parallel interface.
- PCIe speed is classified into lanes, each capable of delivering up to 1GB/s data transfer.
- PCI slots are standardized while PCIe slots vary depending on the number of lanes the slot is intended for.
- Despite PCIe superiority, most manufacturers still use the PCI standard for their devices.

2-6 PCI-X (Peripheral Component Interconnect Extended) Bus

2-6-1 Introduction

PCI-X is a computer bus and expansion card standard that enhances the 32-bit PCI local bus for higher bandwidth demanded mostly by servers and workstations. It uses a modified protocol to support higher clock speeds (up to 133 MHz), but is otherwise similar in electrical implementation. PCI-X 2.0 added speeds up to 533 MHz, with a reduction in electrical signal levels.

2-6-2 Versions

PCI-X 1.0: 64-Bit slots with support for 3.3V and Universal PCI. No support for 5V-only boards. Conventional 33/66 MHz PCI adapters can be used in PCI-X slots. Provides two speed grades: 66 MHz and 133 MHZ.

PCI-X 2.0: Based on PCI-X 1.0. Introduces ECC (Error Correction Codes) mechanism to improve robustness and data integrity. Provides two additional speed grades:

- PCI-X 266: 266 MHz (2.13 GB/sec).
- PCI-X 533: 533 MHz (4.26 GB/sec).

2-6-3 Differences between PCI-X and PCI-E

PCI-X is a 64-bit parallel interface that is backward compatible with 32-bit PCI devices. PCIe is a serial point-to-point connection with a different physical interface that was designed to supersede both PCI and PCI-X.

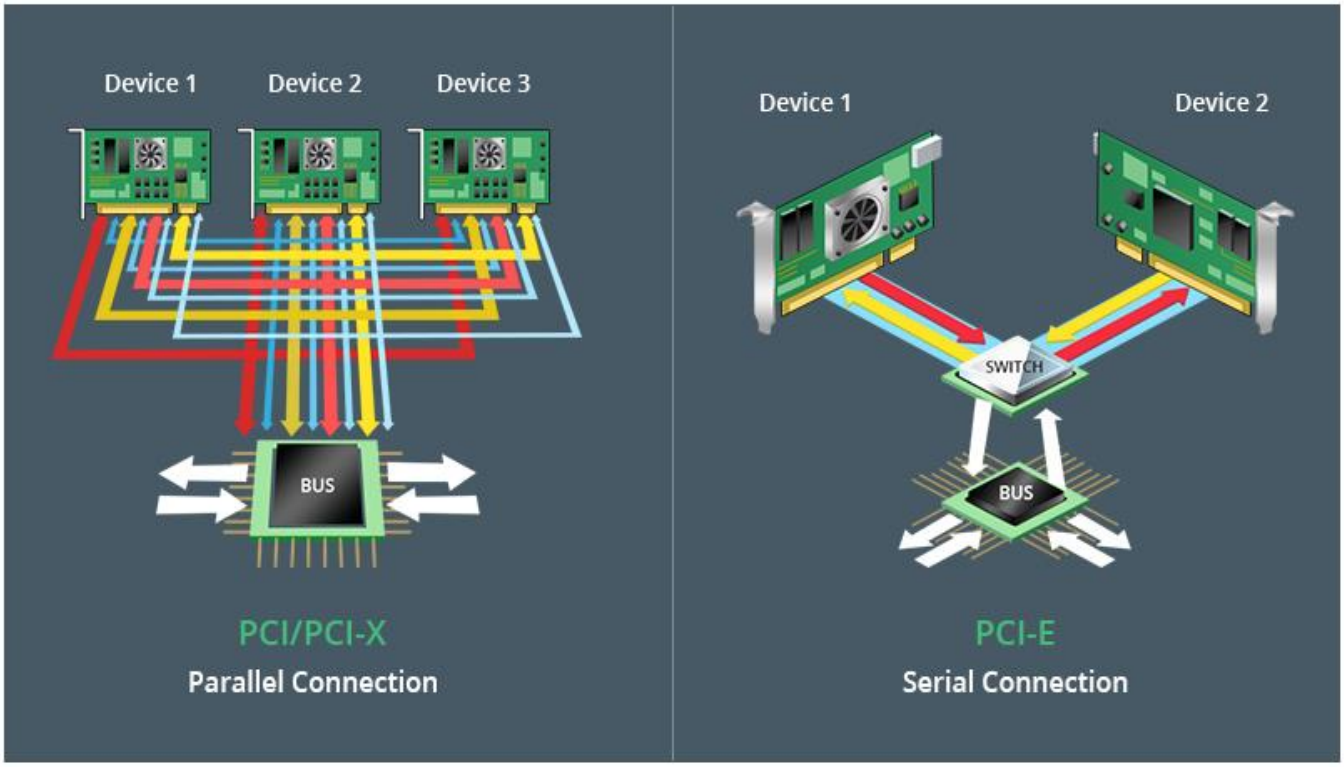


Figure 3: Working Topology of PCI/PCI-X vs PCI-E

As shown in the figure above, the main difference between the previous PCI card and PCI-X card and the successor PCIe card is "parallel vs serial" data transmission. PCI and PCI-X network cards follow the original PCI standard, which is a classic shared bus architecture with all connected peripherals using the same bus in parallel. Specifically, data will be sent and received simultaneously across multiple lines. The devices normally have to wait on the bus when communicating with the computer. The overall performance will go down as the increase of added devices. PCI-E card, however, adopts dedicated point-to-point serial technology, resembling an on-board network. Therefore, each individual device has its own bus, which creates a more efficient bus system. Note that, one serial connection with a higher clock can match the speed of multiple parallel lines moving on the same load.

2-7 CXL (Compute Express Link) Bus

2-7-1 Introduction

Compute Express Link (CXL) is a high-bandwidth, low-latency serial bus interconnect between host processors and devices such as accelerators, memory controllers/buffers, and I/O devices. CXL is based on PCI Express (PCIe) 5.0 physical layer running at 32 GT/s with x16, x8 and x4 link widths. Degraded modes run at 16 GT/s and 8 GT/s with x2 and x1 link widths.

2-7-2 Versions

CXL Specification 1.0: On March 11, 2019, the CXL Specification 1.0 based upon PCIe 5.0 was released. The founding promoter members of the CXL specification included: Alibaba, Cisco, Dell EMC, Facebook, Google, HPE, Huawei, Intel and Microsoft.

CXL Specification 1.1: In June, 2019, the CXL Specification 1.1 was released.

2-7-3 Speed

Table 3: Speed of CXL

Mode type	Link speed	Link width
Native – Primary	32 GT/s	x16
Native – Bifurcation	32 GT/s	x8 and x4
Degraded Modes	32 GT/s	x2 and x1
	16 GT/s	x16, x8, x4, x2 and x1
	8 GT/s	X16, x8, x4, x2 and x1

2-7-4 Protocols

CXL transaction layer consists of three multiplexed sub-protocols that run simultaneously on a single link:

1. **CXL.io** protocol is essentially a PCIe 5.0 protocol with some enhancements and is used for initialization, link-up, device discovery and enumeration, and register access. It provides a non-coherent load/store interface for I/O devices.

2. **CXL.cache** protocol defines interactions between a Host and Device, allowing attached CXL devices to efficiently cache Host memory with extremely low latency using a request and response approach.
3. **CXL.mem** protocol provides a host processor with access to Device-attached memory using load and store commands with the host CPU acting as a master and the CXL Device acting as a subordinate and can support both volatile and persistent memory architectures.

2-7-5 Differences between CXL and PCI-E

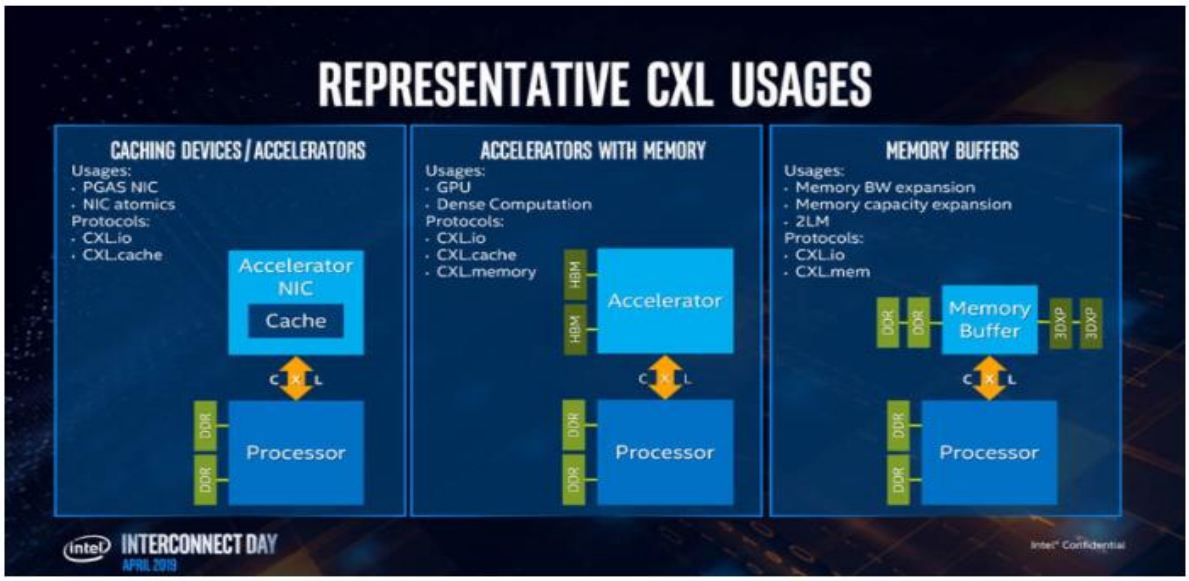


Figure 4: CXL usages

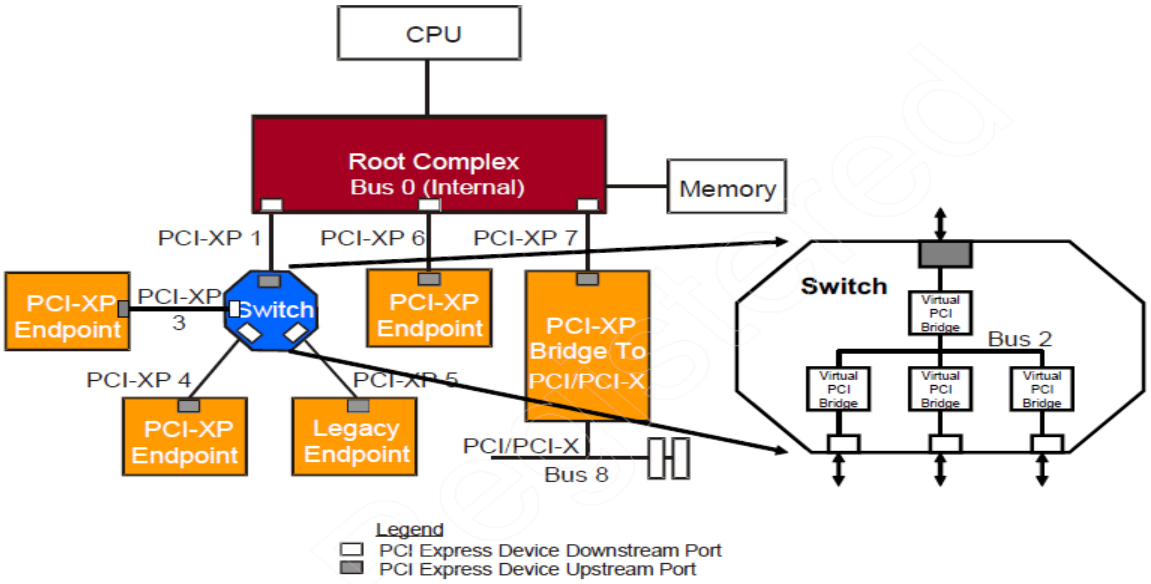


Figure 5: PCI-E Topology

To realize the difference, let's take example. Sharing operands and data between multiple devices, such as two GPU accelerators working on a problem, we will find that

- **PCI-e:** very inefficient.
- **CXL:** efficient.

If we check latency of shared memory pools that span across multiple physical machines, we will find that

- **PCI-e:** High.
- **CXL:** Low.

CXL provides that processor can make transactions directly with memory or accelerator with no need for Root Complex as in PCI-E which is responsible for routing and transmitting all the packets around the hole system.

Chapter 3

Transaction layer

3-1 Introduction

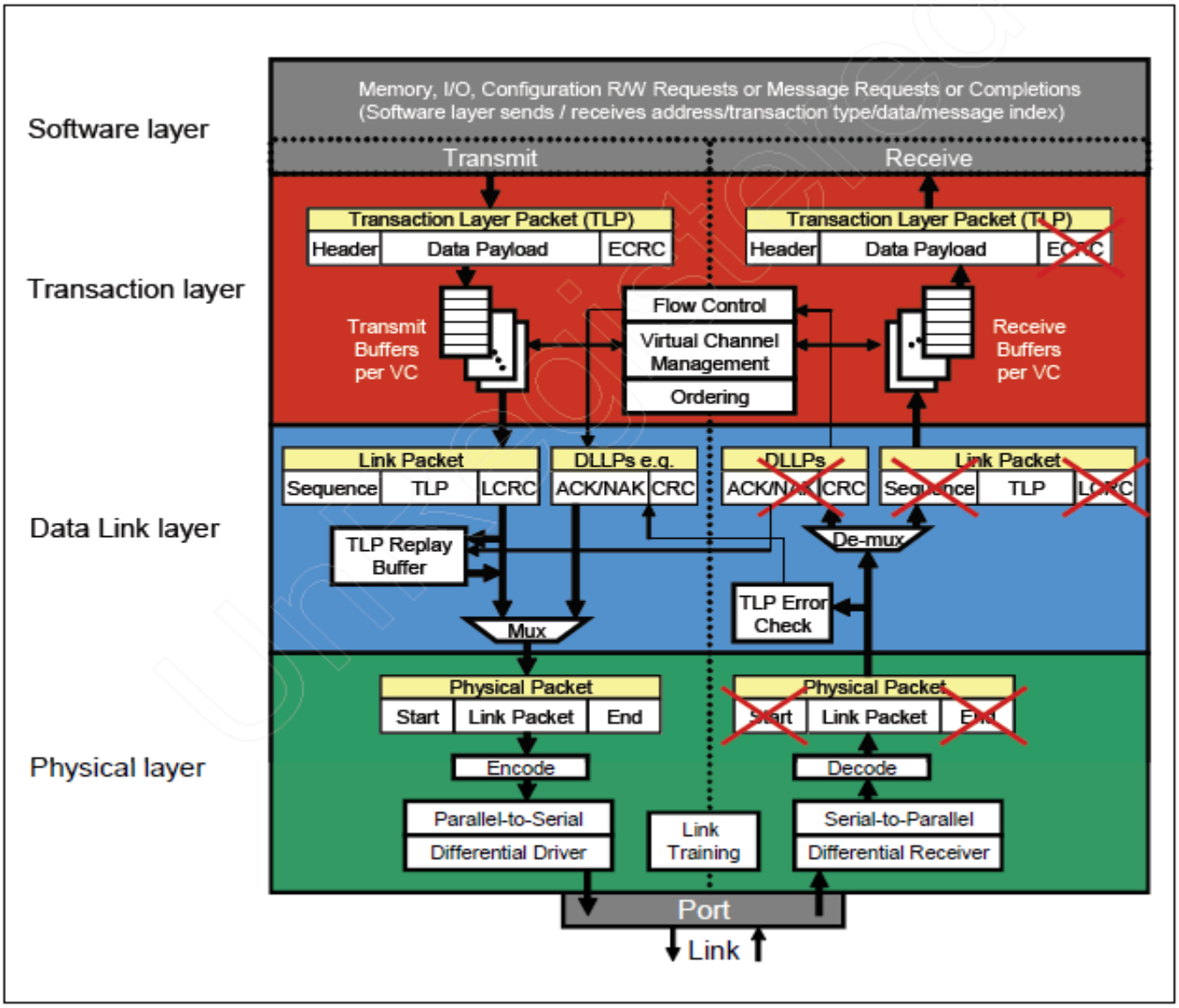


Figure 6: Detailed Block Diagram of PCI Express Device's Layers

The transaction Layer is responsible for generation of out-bound UP traffic and reception of inbound TLP traffic. The Transaction Layer supports the split transaction protocol for non-posted transactions. In other words, the Transaction Layer associates an

inbound completion UP of a given tag value with an outbound non-posted request TLP of the same tag value transmitted earlier.

The transaction layer contains virtual channel buffers (VC Buffers) to store out-bound TLPs that await transmission and also to store inbound TLPs received from the link. The flow control protocol associated with these virtual channel buffers ensures that a remote transmitter does not transmit too many TLPs and cause the receiver virtual channel buffers to overflow. The Transaction Layer also orders TLPs according to ordering rules before transmission. It is this layer that supports the Quality of Service (QoS) protocol. The Transaction Layer supports 4 address spaces: memory address, I0 address, configuration address and message space. Message packets contain a message.

The Transaction Layer receives information from the Device Core and generates outbound request and completion TLPs which it stores in virtual channel buffers. This layer assembles Transaction Layer Packets (TLPs).

The receiver side of the Transaction Layer stores inbound TLPs in receiver virtual channel buffers. The receiver checks for CRC errors based on the ECRC field in the TLP. If there are no errors, the ECRC field is stripped and the resultant information in the TLP header as well as the data payload is sent to the Device Core.

3-2 Project Design for Transaction Layer

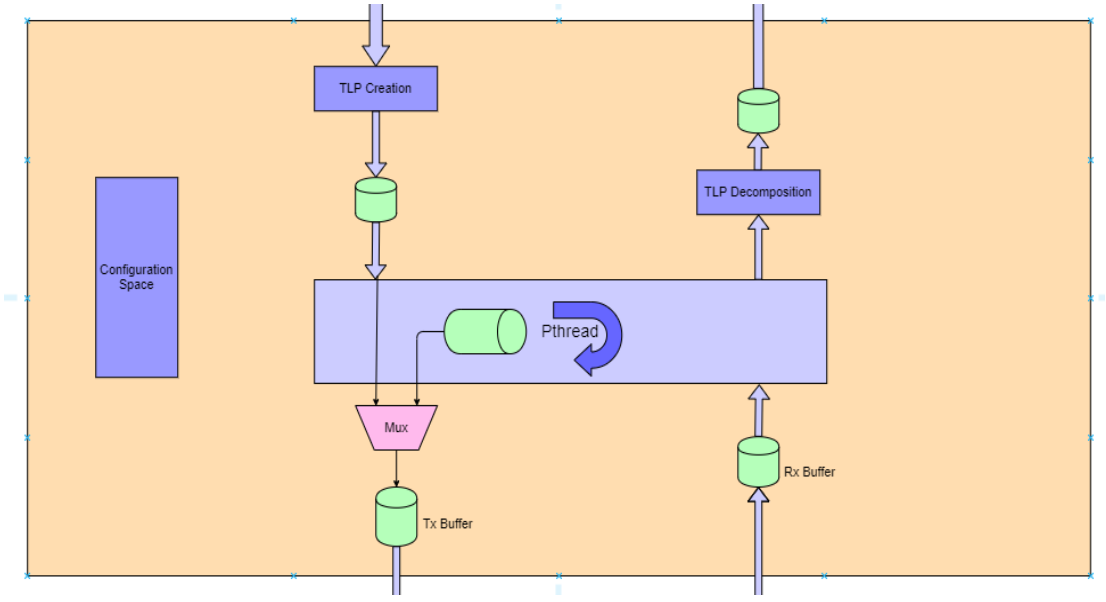


Figure 7: Block Diagram of Transaction Layer

The figure above shows the block diagram of Transaction Layer. In our design, we have implemented the whole layer using C++ in classes, class for the whole layer and each block in the design in also a class (TLP Creation, TLP Decomposition, Configuration Space, Queues and mux) except the P-Thread which is a function in the constructor of the Transaction Layer class.

Transmit Path

The transaction layer takes some information from the application layer to be able to initiate a transaction, this information is address, data and length of data. TLP Creation block takes this information and create the packet, then store it in the queue below it. P-thread block is considered the main block which manages all the transactions and routes them, and it also creates some TLPs and store them in its queue. Both of the queues store their TLPs into Tx Buffer through a mux, then Tx Buffer sends them to the datalink layer.

Receive Path

After the datalink layer check for errors of the packet, it stores the packets into the Rx Buffer, then P-thread routes them to TLP Decomposition block to di-assemble the packet and get the data from it then store the data into the queue above to allow the application layer receiving the data.

3-3 TLP (Transaction Layer Packet) Types

In the table below, we can see all TLP types that can be initiated.

Table 4: TLP Types

TLP Packet Types	Abbreviated Name
Memory Read Request	MRd
Memory Read Request - Locked access	MRdLk
Memory Write Request	MWr
IO Read	IORd
IO Write	IOWr
Configuration Read (Type 0 and Type 1)	CfgRd0, CfgRd1
Configuration Write (Type 0 and Type 1)	CfgWr0, CfgWr1
Message Request without Data	Msg
Message Request with Data	MsgD
Completion without Data	Cpl
Completion with Data	CplD
Completion without Data - associated with Locked Memory Read Requests	CplLk
Completion with Data - associated with Locked Memory Read Requests	CplDLk

In our project, we only support MRd, MWr, CfgRd0, CfgRd1, CfgWr0, CfgWr1, Msg, Cpl and CplD.

3-4 TLP Structure

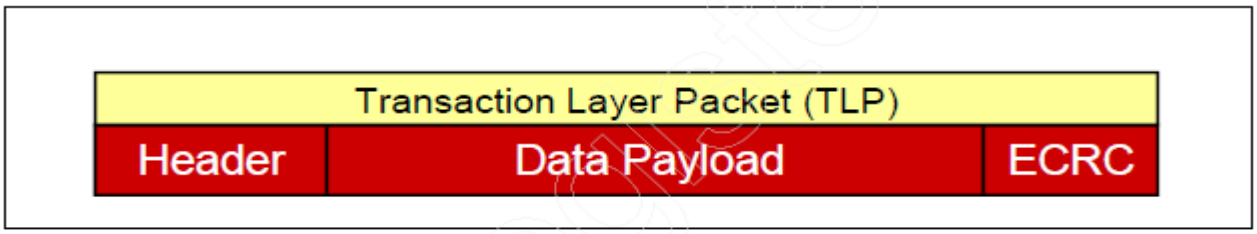


Figure 8: TLP Structure at the Transaction Layer

The major components of a TLP are: Header, Data Payload and an optional ECRC (specification also uses the term Digest) field.

3-4-1 Generic TLP Header

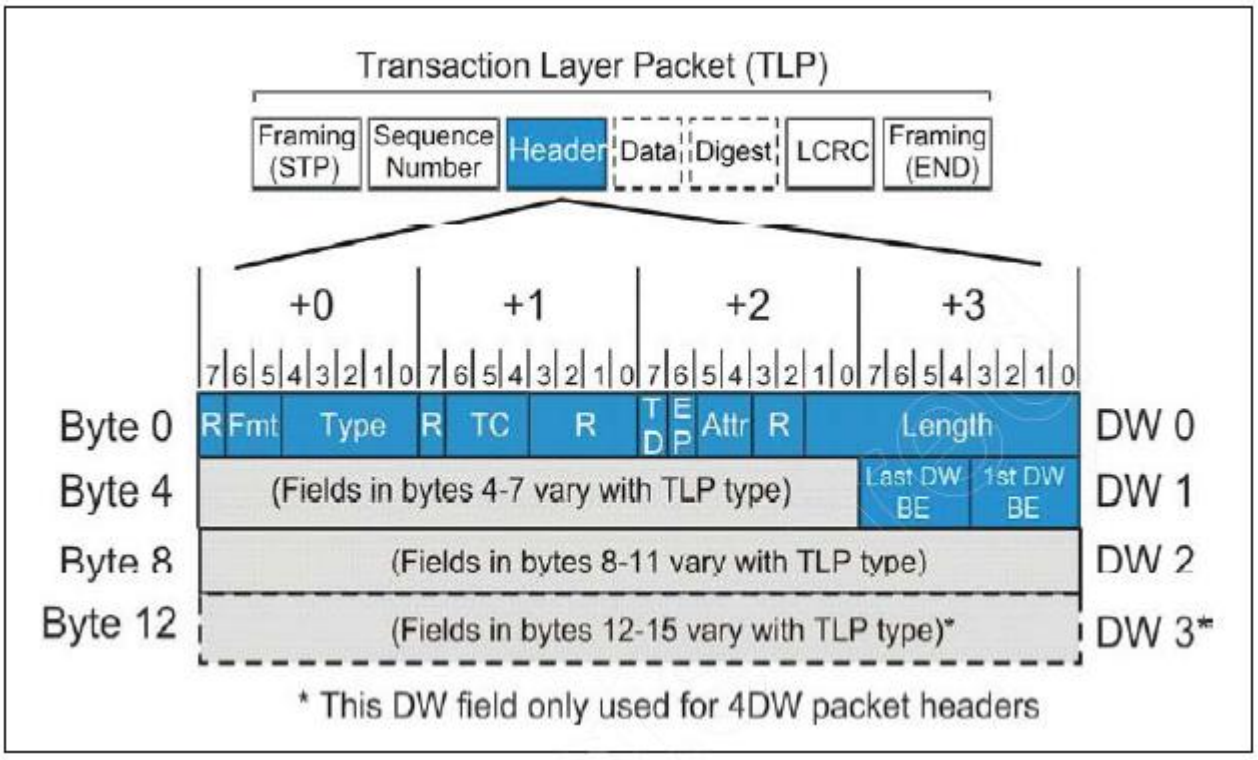


Figure 9: Generic TLP Header Fields

Most of the fields are common in all the TLP types, except some ones which vary with TLP type. Here is summary for each field use:

Table 5: Generic Header Field Summary

Header Field	Header Location	Field Use
Length [9:0]	Byte 3 Bit 7:0 Byte 2 Bit 1:0	TLP data payload transfer size, in DW. Maximum transfer size is 10 bits, 2^{10} = 1024 DW (4KB). Encoding: 00 0000 0001b = 1DW 00 0000 0010b = 2DW . . 11 1111 1111b = 1023 DW 00 0000 0000b = 1024 DW
Attr (Attributes)	Byte 2 Bit 5:4	Bit 5 = Relaxed ordering. When set = 1, PCI-X relaxed ordering is enabled for this TLP. If set = 0, then strict PCI ordering is used. Bit 4 = No Snoop. When set = 1, requester is indicating that no host cache coherency issues exist with respect to this TLP. System hardware is not required to cause processor cache snoop for coherency. When set = 0, PCI -type cache snoop protection is required.
EP (Poisoned Data)	Byte 2 Bit 6	If set = 1, the data accompanying this data should be considered invalid although the transaction is being allowed to complete normally.
TD (TLP Digest Field Present)	Byte 2 Bit 7	If set = 1, the optional 1 DW TLP Digest field is included with this TLP that contains an ECRC value. Some rules: Presence of the Digest field must be checked by all receivers (using this bit). • A TLP with TD = 1, but no Digest field is handled as a Malformed TLP. • If a device supports checking ECRC and TD=1, it must perform the ECRC check. • If a device does not support checking ECRC (optional) at the ultimate destination, the device must ignore the digest.

TC (Traffic Class)	Byte 1 Bit 6:4	These three bits are used to encode the traffic class to be applied to this TLP and to the completion associated with it (if any). 000b = Traffic Class 0 (Default) . . 111b = Traffic Class 7 TC 0 is the default class, and TC 1-7 are used in providing differentiated services. See "Traffic Classes and Virtual Channels" on page 256 for additional information.
Type[4:0]	Byte 0 Bit 4:0	These 5 bits encode the transaction variant used with this TLP. The Type field is used with Fmt [1:0] field to specify transaction type, header size, and whether data payload is present. See below for additional information of Type/Fmt encoding for each transaction type.
Fmt[1:0] Format	Byte 0 Bit 6:5	These two bits encode information about header size and whether a data payload will be part of the TLP: 00b 3DW header, no data 01b 4DW header, no data 10b 3DW header, with data 11b 4DW header, with data See below for additional information of Type/Fmt encoding for each transaction type.
First DW Byte Enables	Byte 7 Bit 3:0	These four high-true bits map one-to-one to the bytes within the first double word of payload. Bit 3 = 1: Byte 3 in first DW is valid; otherwise not Bit 2 = 1: Byte 2 in first DW is valid; otherwise not Bit 1 = 1: Byte 1 in first DW is valid; otherwise not Bit 0 = 1: Byte 0 in first DW is valid; otherwise not See below for details on Byte Enable use.

Last DW Byte Enables	Byte 7 Bit 7:4	These four high-true bits map one-to-one to the bytes within the last double word of payload. Bit 7 = 1: Byte 3 in last DW is valid; otherwise not Bit 6 = 1: Byte 2 in last DW is valid; otherwise not Bit 5 = 1: Byte 1 in last DW is valid; otherwise not Bit 4 = 1: Byte 0 in last DW is valid; otherwise not See below for details on Byte Enable use.

3-4-2 Memory Requests

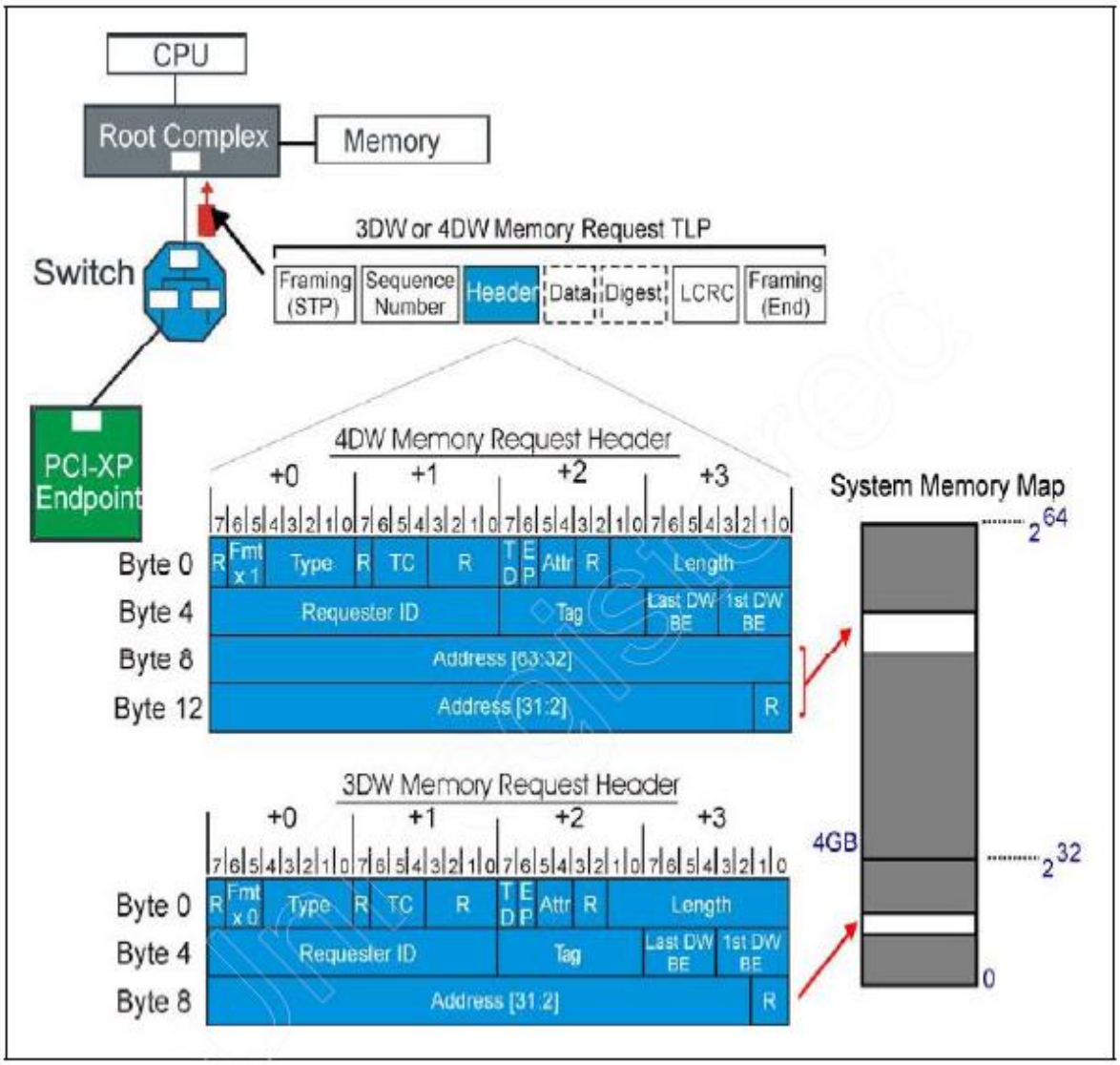


Figure 10: 3D and 4D Memory Request Header Formats

The difference between a 3DW header and a 4DW header is the location and size of the starting Address field:

- For a 3DW header (32-bit addressing): Address bits 31:2 are in Bytes 8-11, and 12-15 are not used.
- For a 4DW header (64-bit addressing): Address bits 31:2 are in Bytes 12-15, and address bits 63:32 are in Bytes 8-11.

Otherwise the header fields are the same.

Table 6: 4DW Memory Request Header Fields

Field Name	Header Byte/Bit	Function
Length [9:0]	Byte 3 Bit 7:0 Byte 2 Bit 1:0	TLP data payload transfer size, in DW. Maximum transfer size is 10 bits, 2^{10} = 1024 DW (4KB). Encoding: 00 0000 0001b = 1DW 00 0000 0010b = 2DW . . 11 1111 1111b = 1023 DW 00 0000 0000b = 1024 DW
Attr (Attributes)	Byte 2 Bit 5:4	Bit 5 = Relaxed ordering. When set = 1, PCI-X relaxed ordering is enabled for this TLP. If set = 0, then strict PCI ordering is used. Bit 4 = No Snoop. When set = 1, requester is indicating that no host cache coherency issues exist with respect to this TLP. System hardware is not required to cause processor cache snoop for coherency. When set = 0, PCI -type cache snoop protection is required.
EP (Poisoned Data)	Byte 2 Bit 6	If set = 1, the data accompanying this data should be considered invalid although the transaction is being allowed to complete normally.
TD (TLP Digest Field Present)	Byte 2 Bit 7	If set = 1, the optional 1 DW TLP Digest field is included with this TLP. Some rules: Presence of the Digest field must be checked by all receivers (using this bit) • A TLP with TD = 1, but no Digest field is handled as a Malformed TLP. • If a device supports checking ECRC and TD=1, it must perform the ECRC check. • If a device does not support checking ECRC (optional) at the ultimate destination, the device must ignore the digest field.
TC (Traffic Class)	Byte 1 Bit 6:4	These three bits are used to encode the traffic class to be applied to this TLP and to the completion associated with it (if any). 000b = Traffic Class 0 (Default) . . 111b = Traffic Class 7 TC 0 is the default class, and TC 1-7 are used in providing differentiated services. See"Traffic Classes and Virtual Channels" on page 256 for additional information.

Type[4:0]	Byte 0 Bit 4:0	TLP packet Type field: 00000b = Memory Read or Write 00001b = Memory Read Locked Type field is used with Fmt [1:0] field to specify transaction type, header size, and whether data payload is present.
Fmt 1:0 (Format)	Byte 0 Bit 6:5	Packet Format: 00b = Memory Read (3DW w/o data) 10b = Memory Write (3DW w/ data) 01b = Memory Read (4DW w/o data) 11b = Memory Write (4DW w/ data)
1st DW BE 3:0 (First DW Byte Enables)	Byte 7 Bit 3:0	These high true bits map one-to-one to qualify bytes within the DW pay-load.
Last BE 3:0 (Last DW Byte Enables)	Byte 7 Bit 7:4	These high true bits map one-to-one to qualify bytes within the last DW transferred.
Tag 7:0	Byte 6 Bit 7:0	These bits are used to identify each outstanding request issued by the requester. As non-posted requests are sent, the next sequential tag is assigned. Default: only bits 4:0 are used (32 out-standing transactions at a time) If Extended Tag bit in PCI Express Control Register is set = 1, then all 8 bits may be used (256 tags).
Requester ID 15:0	Byte 5 Bit 7:0 Byte 4 Bit 7:0	Identifies the requester so a comple-tion may be returned, etc. Byte 4, 7:0 = Bus Number Byte 5, 7:3 = Device Number Byte 5, 2:0 = Function Number
Address 31:2	Byte 15 Bit 7:2 Byte 14 Bit 7:0 Byte 13 Bit 7:0 Byte 12 Bit 7:0	The lower 32 bits of the 64 bit start address for the memory transfer. Note that the lower two bits of the 32 bit address are reserved (00b), forcing the start address to be DW aligned.
Address 63:32	Byte 11 Bit 7:2 Byte 10 Bit 7:0 Byte 9 Bit 7:0 Byte 8 Bit 7:0	The upper 32 bits of the 64-bit start address for the memory transfer.

Memory Request Notes. Features of memory requests include:

1. Memory transfers are never permitted to cross a 410 boundary.
2. All memory mapped writes are posted, resulting in much higher performance.
3. Either 32 bit or 64-bit addressing may be used. The 3DW header format sup-ports 32-bit addresses and the 4DW header supports 64 bits.
4. The full capability of burst transfers is available with a transfer length of 0-1024 DW (0-4KB).

5. Advanced PCI Express Quality of Service features, including up to 8 transfer classes and virtual channels may be implemented.
6. The No Snoop attribute bit in the header may be set = 1, relieving the system hardware from the burden of snooping processor caches when PCI Express transactions target main memory. Optionally, the bit may be de-asserted in the packet, providing PCI-like cache coherency protection.
7. The Relaxed Ordering bit may also be set = 1, permitting devices in the path between the packet and its destination to apply the relaxed ordering rules available in PCI-X. If de-asserted, strong PCI producer-consumer ordering is enforced.

3-4-3 Configuration Requests

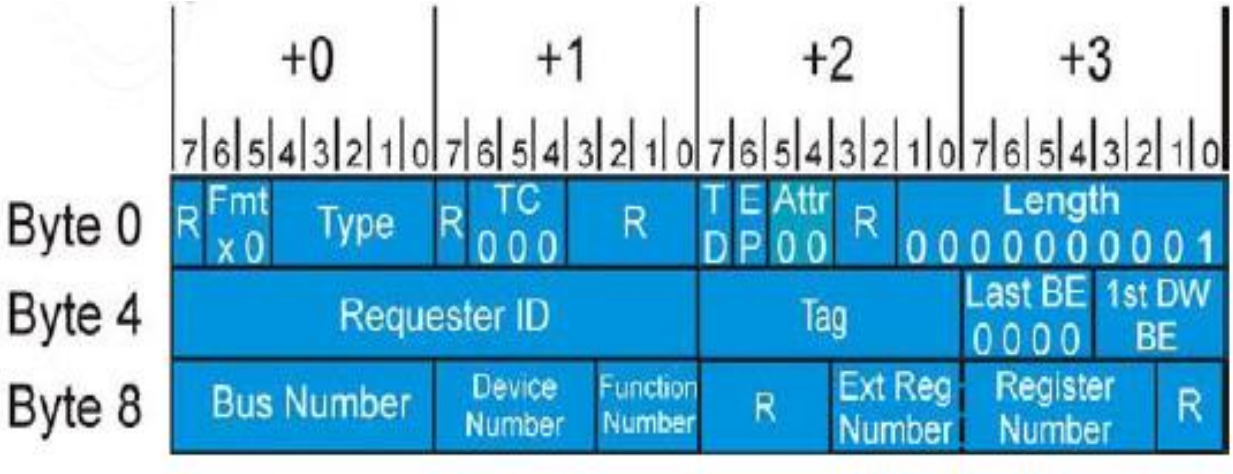


Figure 11: 3DW Configuration Request Header Format

To maintain compatibility with PCI, PCI Express supports both Type 0 and Type 1 configuration cycles. A Type 1 cycle propagates downstream until it reaches the bridge interface hosting the bus (link) that the target device resides on. The configuration transaction is converted on the destination link from Type 1 to Type 0 by the bridge. The bridge forwards and converts configuration cycles using previously programmed Bus Number registers that specify its primary, secondary and subordinate buses.

Table 7: Configuration Request Header Fields

Field Name	Header Byte/Bit	Function
Length 9:0	Byte 3 Bit 7:0 Byte 2 Bit 1:0	Indicates data payload size in DW. For configuration requests, this field is always = 1. Byte Enables are used to qualify bytes within DW (any combination is legal)
Attr 1:0 (Attributes)	Byte 2 Bit 5:4	Attribute 1: Relaxed Ordering Bit Attribute 0: No Snoop Bit Both of these bits are always = 0 in configuration requests.
EP	Byte 2 Bit 6	If = 1, indicates the data payload (if present) is poisoned.
TD	Byte 2 Bit 7	If = 1, indicates the presence of a digest field (1 DW) at the end of the TLP (preceding LCRC and END)
TC 2:0 (Transfer Class)	Byte 2 Bit 6:4	Indicates transfer class for the packet. TC is = 0 for all Configuration requests.
Type 4:0	Byte 0 Bit 4:0	TLP packet type field. Set to: 00100b = Type 0 config request 00101b = Type 1 config request
Fmt 1:0 (Format)	Byte 0 Bit 6:5	Packet Format. Always a 3DW header 00b = configuration read (no data) 10b = configuration write (with data)
1st DW BE 3:0 (First DW Byte Enables)	Byte 7 Bit 3:0	These high true bits map one-to-one to qualify bytes within the DW payload. For config requests, any bit combination is valid (including none)

Last BE 3:0 (Last DW Byte Enables)	Byte 7 Bit 7:4	These high true bits map one-to-one to qualify bytes within the last DW transferred. For config requests, these bits must be 0000b. (Single DW)
Tag 7:0	Byte 6 Bit 7:0	These bits are used to identify each outstanding request issued by the requester. As non-posted requests are sent, the next sequential tag is assigned. Default: only bits 4:0 are used (32 outstanding transactions at a time) If Extended Tag bit in PCI Express Control Register is set = 1, then all 8 bits may be used (256 tags).
Requester ID 15:0	Byte 5 Bit 7:0 Byte 4 Bit 7:0	Identifies the requester so a completion may be returned, etc. Byte 4, 7:0 = Bus Number Byte 5, 7:3 = Device Number Byte 5, 2:0 = Function Number
Register Number	Byte 11 Bit 7:2	These bits provide the lower 6 bits of DW configuration space offset. The Register Number is used in conjunction with Ext Register Number to provide the full 10 bits of offset needed for the 1024 DW (4096 byte) PCI Express configuration space.
Ext Register Number (Extended Register Number)	Byte 10 Bit 3:0	These bits provide the upper 4 bits of DW configuration space offset. The Ext Register Number is used in conjunction with Register Number to provide the full 10 bits of offset needed for the 1024 DW (4096 byte) PCI Express configuration space. For compatibility, this field can be set = 0, and only the lower 64DW (256 bytes will be seen) when indexing the Register Number.
Completer ID 15:0	Byte 9 Bit 7:0 Byte 8 Bit 7:0	Identifies the completer being accessed with this configuration cycle. The Bus and Device numbers in this field are "captured" by the device on each configuration Type 0 write. Byte 8, 7:0 = Bus Number Byte 9, 7:3 = Device Number Byte 9, 2:0 = Function Number

Configuration Request Notes. Configuration requests always use the 3DW header format and are routed by the contents of the ID field.

All devices “capture” the Bus Number and Device Number information provided by the upstream device during each Type 0 configuration write cycle. Information is contained in Completer ID.

3-4-4 Completion Requests

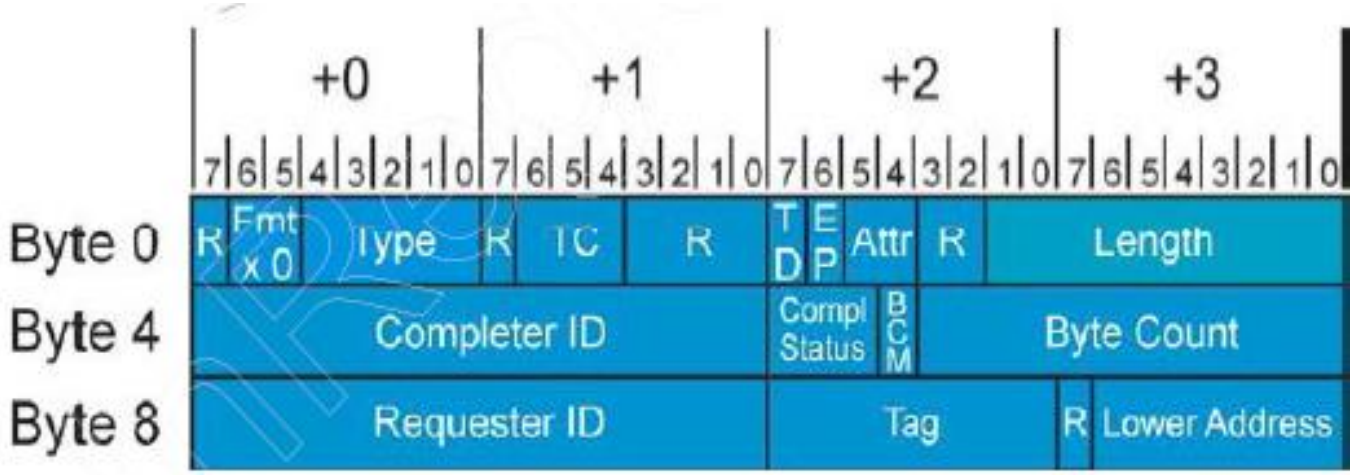


Figure 12: Completion Request Header Format

Many of the fields in the completion must have the same values as the associated request, including Traffic Class, Attribute bits, and the original Requester ID which is used to route the completion back to the original requester.

Table 8: Completion Request Header Fields

Field Name	Header Byte/Bit	Function
Length 9:0	Byte 3 Bit 7:0 Byte 2 Bit 1:0	Indicates data payload size in DW. For completions, this field reflects the size of the data payload associated with this completion.
Attr 1:0 (Attributes)	Byte 2 Bit 5:4	Attribute 1: Relaxed Ordering Bit Attribute 0: No Snoop Bit For a completion, both of these bits are set to same state as in the request.
EP	Byte 2 Bit 6	If = 1, indicates the data payload is poi-soned.
TD	Byte 2 Bit 7	If = 1, indicates the presence of a digest field (1 DW) at the end of the TLP (preceding LCRC and END)
TC 2:0 (Transfer Class)	Byte 2 Bit 6:4	Indicates transfer class for the packet. For a completion, TC is set to same value as in the request.
Type 4:0	Byte 0 Bit 4:0	TLP packet type field. Always set to 01010b for a completion.
Fmt 1:0 (Format)	Byte 0 Bit 6:5	Packet Format. Always a 3DW header 00b = Completion without data (Cpl) 10b = Completion with data (CplD)
Byte Count	Byte 7 Bit 7:0 Byte 6 Bit 3:0	This is the remaining byte count until a read request is satisfied. Generally, it is derived from the original request Length field. See "Data Returned For Read Requests:" on page 188 for special cases caused by multiple completions.

BCM (Byte Count Modified)	Byte 6 Bit 4	Set = 1 only by PCI-X completers. Indicates that the byte count field (see previous field) reflects the first transfer payload rather than total payload remaining. See "Using The Byte Count Modified Bit" on page 188.
CS 2:0 (Completion Status Code)	Byte 6 Bit 7:5	These bits encoded by the completer to indicate success in fulfilling the request. 000b = Successful Completion (SC) 001b = Unsupported Request (UR) 010b = Config Req Retry Status (CR S) 100b = Completer abort. (CA) others: reserved. See "Summary of Completion Status Codes:" on page 187.
Completer ID 15:0	Byte 5 Bit 7:0 Byte 4 Bit 7:0	Identifies the completer. While not needed for routing a completion, this information may be useful if debugging bus traffic. Byte 4 7:0 = Completer Bus # Byte 5 7:3 = Completer Dev # Byte 5 2:0 = Completer Function #
Lower Address 6:0	Byte 11 Bit 6:0	The lower 7 bits of address for the first enabled byte of data returned with a read. Calculated from request Length and Byte enables, it is used to determine next legal Read Completion Boundary. See "Calculating Lower Address Field" on page 187.
Tag 7:0	Byte 10 Bit 7:0	These bits are set to reflect the Tag received with the request. The requester uses them to associate inbound completion with an outstanding request.

Requester ID 15:0	Byte 9 Bit 7:0 Byte 8 Bit 7:0	Copied from the request into this field to be used in routing the completion back to the original requester. Byte 4, 7:0 = Requester Bus # Byte 5, 7:3 = Requester Device # Byte 5, 2:0 = Requester Function #

Summary of Completion Status Codes

- 000b (SC) Successful Completion code indicates the original request completed properly at the target.
- 001b (UR) Unsupported Request code indicates original request failed at the target because it targeted an unsupported address, carried an unsupported address or request, etc. This is handled as an uncorrectable error.
- 010b (CRS) Configuration Request Retry Status indicates target was temporarily off-line and the attempt should be retried. (e.g. initialization delay after reset, etc.).
- 100b (CA) Completer Abort code indicates that completer is off-line due to an error (much like target abort in PCI). The error will be logged and handled as uncorrectable error.

3-4-5 Message Requests

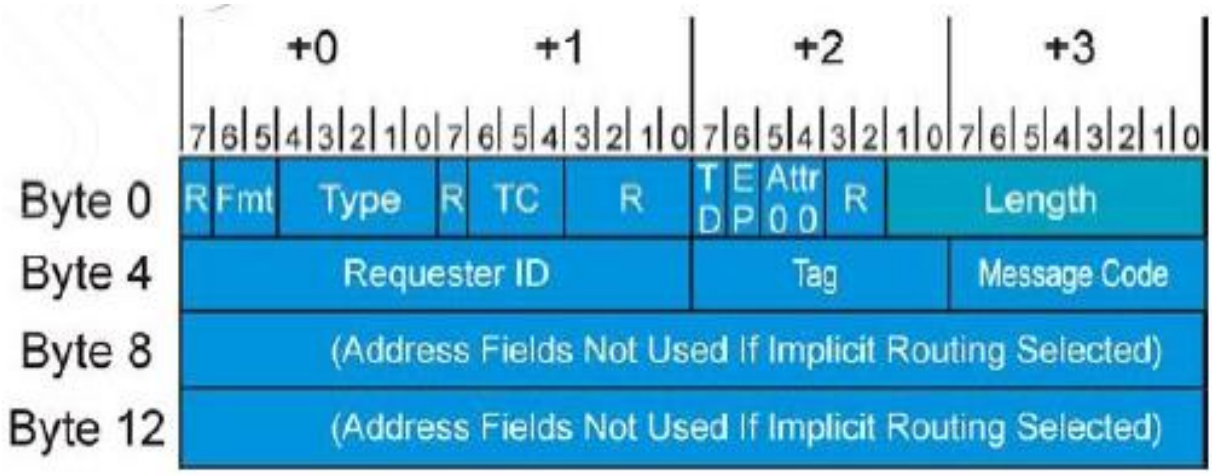


Figure 13: Message Request Header Format

Message requests replace many of the interrupt, error, and power management sideband signals used on earlier bus protocols. All message requests use the 4DW header format, and are handled much the same as posted memory write transactions. Messages may be routed using address, ID, or implicit routing. The muting subfield in the packet header indicates the routing method to apply, and which additional header registers are in use (address registers, etc.).

Table 9: Message Request Header Fields

Field Name	Header Byte/Bit	Function
Length 9:0	Byte 3 Bit 7:0 Byte 2 Bit 1:0	Indicates data payload size in DW. For message requests, this field is always 0 (no data) or 1 (one DW of data)
Attr 1:0 (Attributes)	Byte 2 Bit 5:4	Attribute 1: Relaxed Ordering Bit Attribute 0: No Snoop Bit Both of these bits are always = 0 in message requests.
EP	Byte 2 Bit 6	If = 1, indicates the data payload (if present) is poisoned.
TD	Byte 2 Bit 7	If = 1, indicates the presence of a digest field (1 DW) at the end of the TLP (preceding LCRC and END)
TC 2:0 (Transfer Class)	Byte 2 Bit 6:4	Indicates transfer class for the packet. TC is = 0 for all message requests.
Type 4:0	Byte 0 Bit 4:0	TLP packet type field. Set to: Bit 4:3: 10b = Msg Bit 2:0 (Message Routing Subfield) 000b = Routed to Root Complex 001b = Routed by address 010b = Routed by ID 011b = Root Complex Broadcast Msg 100b = Local; terminate at receiver 101b = Gather/route to Root Complex 0thers = reserved
Fmt 1:0 (Format)	Byte 0 Bit 6:5	Packet Format. Always a 4DW header 01b = message request without data 11b = message request with data

Message Code 7:0	Byte 7 Bit 7:0	This field contains the code indicating the type of message being sent. 0000 0000b = Unlock Message 0001 xxxxb = Power Mgmt Message 0010 0xxxb = INTx Message 0011 00xxb = Error Message 0100 xxxxb = Hot Plug Message 0101 0000b = Slot Power Message 0111 111xb = Vendor Type 0 Message 0111 1111b = Vendor Type 1 Message
Tag 7:0	Byte 6 Bit 7:0	As all message requests are posted, no tag is assigned to them. These bits should be = 0.
Requester ID 15:0	Byte 5 Bit 7:0 Byte 4 Bit 7:0	Identifies the requester sending the message. Byte 4, 7:0 = Requester Bus # Byte 5, 7:3 = Requester Device # Byte 5, 2:0 = Requester Function #
Address 31:2	Byte 11 Bit 7:2 Byte 10 Bit 7:0 Byte 9 Bit 7:0 Byte 8 Bit 7:0	If address routing was selected for the message (see Type 4:0 field above), then this field contains the lower part of the 64-bit starting address. Otherwise, this field is not used.
Address 63:32	Byte 15 Bit 7:2 Byte 14 Bit 7:0 Byte 13 Bit 7:0 Byte 12 Bit 7:0	If address routing was selected for the message (see Type 4:0 field above), then this field contains the upper 32 bits of the 64 bit starting address. Otherwise, this field is not used.

3-5 Configuration Space

3-5-1 Introduction

Configuration is a set of registers used to Identify the device and control its functionality and sense its status in a generic manner.it has two types Type 0 and Type 1. Type 0 is used in endpoints but type 1 used in Switches and bridges which has some different registers than Type 0 as its used to control a bridge.

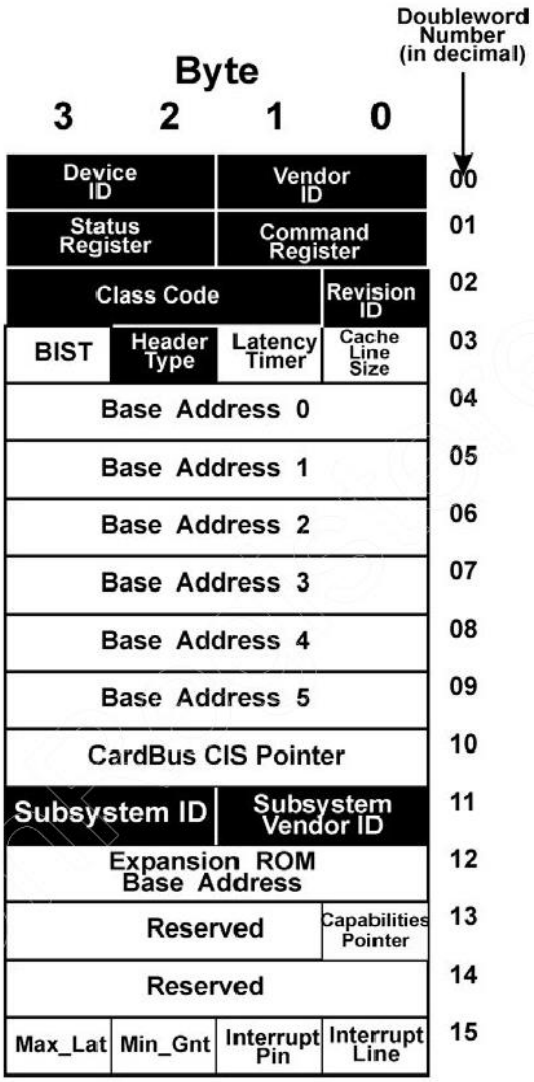


Figure 14:Configuration Space Type 0

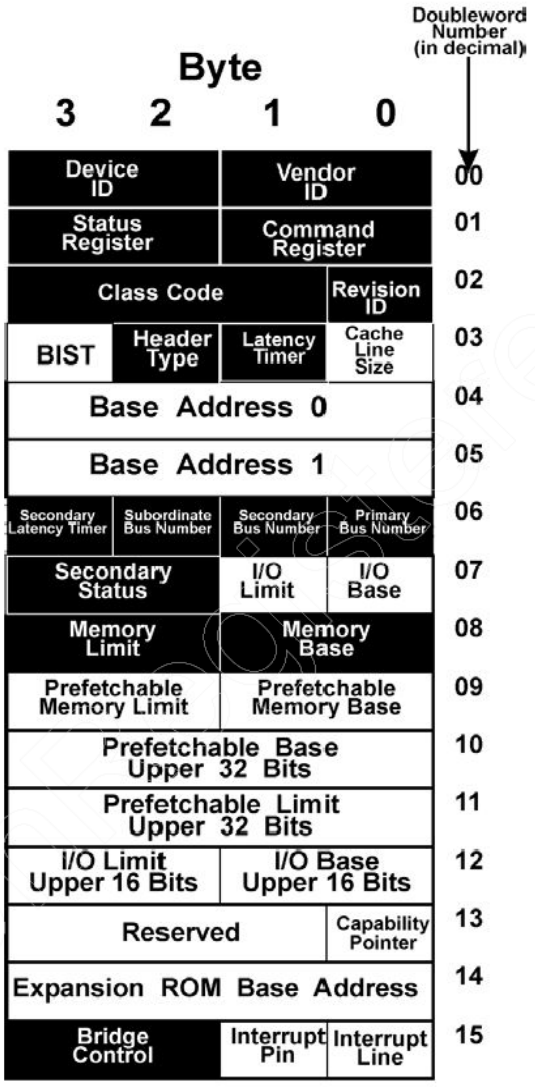


Figure 15: Configuration Space Type 1

3-5-2 Type 0

Its 16 DW Configuration Space used for a lot of functions. But not all of them is for PCIe some of these registers are read only and hardwired to zero: -

- Latency Timer
- Cache Line Size
- Max_Lat
- Min_Gnt

Registers used to identify device's driver: -

OS uses combination of registers to determine which driver to load for the device.

- Vendor ID
- Device ID
- Revision ID
- Class Code
- SubSystem Vendor ID
- SubSystem ID

Vendor ID

16-bit Register identifies the manufacturer of the function. The value hardwired in this read only register is assigned by a central authority (The PCI SIG) that controls issuance of the number. The value FFFFh is reserved and must be returned by the Host/PCI Bridge when an attempt is made to perform a configuration read from a configuration register in a non-existent function.

Device ID

16-bit Register Assigned by the manufacturer of the function and identifies the type of the function. In conjunction with the vendor ID and Possibly the Revision ID. The Device ID can be used to locate a function specific driver.

Revision ID

8-bit Register Assigned by the manufacturer of the function and identifies the Revision number of the function. If the vendor has supplied a revision-specific driver, this is handy in ensuring that the correct driver is loaded by the OS.

Class Code

The Class Code register is a 24-bit, read-only register divided into three fields: base Class, Sub Class, and Programming Interface. It identifies the basic function of the function (e.g., a mass storage controller), a more specific function sub-class (e.g., IDE mass storage controller), and, in some cases, a register-specific programming interface (such as a specific flavor of the IDE register set)

- The upper byte defines the base Class of the function
- The middle byte defines a sub-class within the base Class
- The lower byte defines the Programming Interface.

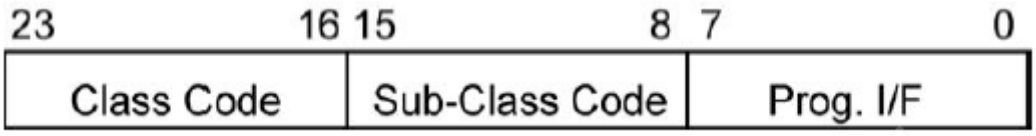


Figure 16: Class Code Register

The currently-defined base Class codes are listed in the Second Table. First Table define example for some Subclasses within the base Class. For many Class/Subclass categories, the Programming Interface byte is hardwired to return zeros (in other words, it has no meaning). For some, such as VGA-compatible functions and IDE controllers, it does have meaning.

This register is useful when the OS is attempting to locate a function that a Class driver can work with. As an example, assume that a particular device driver has been written to work with any display adapter that is 100% XGA register set compatible. If the OS can locate a function with a Class of 03h and a Sub Class of 0lh, the driver will work with that function. A Class driver is more flexible than a driver that has been written to work only with a specific function from a specific vendor.

The Programming Interface Byte. For some functions (such as the XGA display adapter used as an example in the previous section) the combination of the Class Code and Sub Class Code is sufficient to fully-define its level of register set compatibility. The register set layout for some function types, however, can vary from one implementation to another. As an example, from a programming interface perspective there are a number of flavors of IDE mass storage controllers, so it's not sufficient to identify yourself as an IDE mass storage controller. The Programming Interface byte value provides the final level of granularity that identifies the exact register set layout of the function.

Sub-Class	Prog. I/F	Description
02h	00h	Floppy disk controller.
03h	00h	IPI controller.
04h	00h	RAID controller.
05h	20h	ATA controller with single DMA .
	30h	ATA controller with chained DMA.
80h	00h	Other mass storage controller.

Table 10: mass storage subclasses example

Class	Description
00h	Function built before class codes were defined (in other words: before rev 2.0 of the PCI spec).
01h	Mass storage controller.
02h	Network controller.
03h	Display controller.
04h	Multimedia device.
05h	Memory controller.
06h	Bridge device.
07h	Simple communications controllers.
08h	Base system peripherals.
09h	Input devices.
0Ah	Docking stations.
0Bh	Processors.
0Ch	Serial bus controllers.
0Dh	Wireless controllers.
0Eh	Intelligent IO controllers.
0Fh	Satellite communications controllers.
10h	Encryption/Decryption controllers.
11h	Data acquisition and signal processing controllers.
12h-FEh	Reserved.
FFh	Device does not fit any of the defined class codes.

Table 11: Class Codes

Subsystem Vendor ID & Subsystem ID

The Subsystem Vendor ID is obtained from the SIG, while the vendor supplies its own Subsystem ID (the full name of this register is really "Sub-system Device ID", but the "device" is silent). A value of zero in these registers indicates there isn't a Subsystem Vendor and Subsystem ID associated with the function.

These two mandatory registers (Subsystem Vendor ID and Subsystem ID) are used to uniquely identify the add-in card or subsystem that the function resides within. Using these two registers, the OS can distinguish the difference between cards or subsystems manufactured by different vendors but designed around the same third-party core logic. This permits the Plug-and-Play OS to locate the correct driver to load into memory.

Must Contain Valid Data When First Accessed. These two registers must contain their assigned values before the system first accesses them. If software attempts to access them before they have been initialized, the device must issue:

- a Retry to the master (in PCI).
- a Completion with CRS (Configuration Request Retry Completion Status) in PCI Express.

The values in these registers could be hardwired, loaded from a serial EEPROM, determined from hardware strapping pins, etc.

Header Type

Bits [6:0] of this one-byte register define the format of DW 4-through-15 of the function's configuration Header. In addition, bit seven defines the device as a single (bit 7 = 0) or multifunction (bit 7 =1) device. During configuration, the programmer determines if there are any other functions in this device that require configuration by testing the state of bit seven.

Currently, the only Header formats defined other than Header Type Zero are:

- Header Type One (PCI-to-PCI bridge Header format).
- Header Type Two (CardBus bridge).

Future versions of the specification may define other formats.

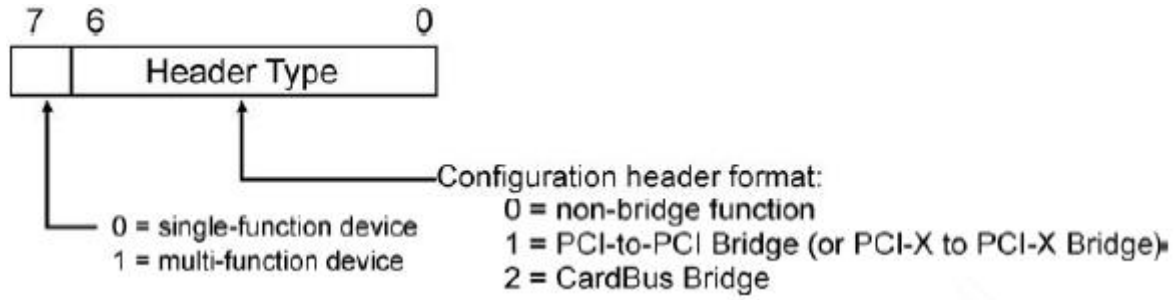


Figure 17: Header Type Register

BIST

This Optional register may be implemented by both Requester and Completer functions. If a function implements a Built-In Self-Test (BIST), If the function doesn't support a BIST, this register must return zeros when read. The function's BIST is invoked by setting bit six to one. The function resets bit six upon completion of the BIST. Configuration software must fail the function if it doesn't reset bit six within two seconds. At the conclusion of the BIST, the test

result is indicated in the lower four bits of the register. A completion code of zero indicates successful completion. A non-zero value represents a function-specific error code.

The time limit of two seconds may not be sufficient time to test a very complex function or one with an extremely large buffer that needs to be tested. In that case, the remainder of the test could be completed in the initialization portion of the function's device driver when the OS loads it into memory and calls it.

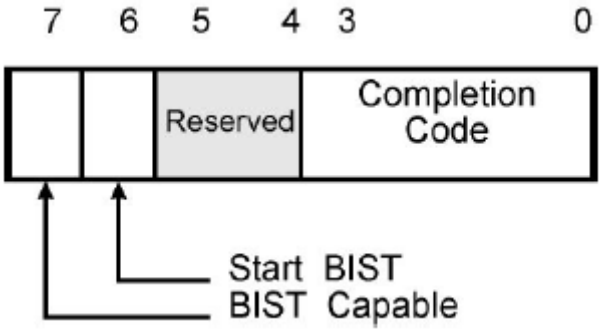


Figure 18: BIST Register

Bit	Function
3:0	**Completion Code**. A value of zero indicates successful completion, while a non-zero result indicates a function-specific error.
5:4	Reserved.
6	**Start BIST**. Writing a one into this bit starts the function's BIST. The function resets this bit automatically upon completion. Software should fail the function if the BIST does not complete within two seconds.
7	**BIST Capable**. Should return a one if the function implements a BIST, a zero if it doesn't.

Table 12: BIST Register Bit assignment

Capabilities Pointer

Configuration Header Space Not Large Enough

The 2.1 PCI spec defined the first 16 DWs of a function's PCI-compatible configuration space as its configuration Header space. It was originally intended that all of the function's PCI spec-defined configuration registers would reside within this region and that all of its function-specific configuration registers would reside within the lower 48 DWs of its PCI-compatible configuration space. Unfortunately, they ran out of space when defining new configuration registers in the 2.2 PCI spec. For this reason, the 2.2 and 2.3 PCI specs permit some spec-defined registers to be implemented in the lower 48 DWs of a function's PCI-compatible configuration space.

Discovering That Capabilities Exist

If the Capabilities List bit in the Status register is set to one, the function implements the Capabilities Pointer register in byte zero of DW 13 in its PCI-compatible configuration space. This implies that the pointer contains the DW-aligned start address of the Capabilities List within the function's lower 48 DWs of PCI-compatible configuration space. It is a rule that the two least-significant bits must be hard-wired to zero and must be ignored (i.e., masked) by software when reading the register. The upper six bits represents the upper six bits of the 8-bit, DW-aligned start address of the new registers implemented in the lower 48 DWs of the function's PCI-compatible space. The two least-significant bits are assumed to be zero.

What the Capabilities List Looks Like

The configuration location pointed to by the Capabilities Pointer register is the first entry in a linked series of one or more configuration register sets, each of which supports a feature. Each entry has the general format illustrated in Figure 22-6 on page 782. The first byte is referred to as the Capability ID (assigned by the PCI SIG) and identifies the feature associated with this register set (e.g., 2 = AGP), while the second byte either points to another feature's register set, or indicates that there are no additional register sets (with a pointer value of zero) associated with this function. In either case, the least-significant two bits must return zero. If a pointer to the next feature's register set is present in the second byte, it points to a DW within the function's lower 48 DWs of PCI-compatible configuration space (it can point either forward or backward in the function's configuration space). The respective feature's register set always immediately follows the first two bytes of the entry, and its length and format are defined by what type of feature it is. The Capabilities currently defined in the 2.3 PCI spec are those listed in Currently-Assigned Capabilities ID Table.

Figure 19: General Format of a New Capabilities List Entry

ID	Description
00h	Reserved.
01h	**PCI Power Management Interface**. Refer to "The PM Capability Register Set" on page 585.
02h	**AGP**. Refer to "AGP Capability" on page 845. Also refer to the MindShare book entitled *AGP System Architecture, Second Edition* (published by Addison-Wesley).
03h	**VPD**. Refer to "Vital Product Data (VPD) Capability" on page 848.
04h	**Slot Identification**. This capability identifies a bridge that provides external expansion capabilities (i.e., an expansion chassis containing add-in card slots). Full documentation of this feature can be found in the revision 1.1 *PCI-to-PCI Bridge Architecture Specification*. For a detailed, Express-oriented description, refer to "Introduction To Chassis/Slot Numbering Registers" on page 859 and "Chassis and Slot Number Assignment" on page 861.
05h	**Message Signaled Interrupts**. Refer to "The MSI Capability Register Set" on page 332.
06h	**CompactPCI Hot Swap**. Refer to the chapter entitled *Compact PCI and PMC* in the MindShare book entitled *PCI System Architecture, Fourth Edition* (published by Addison-Wesley).
07h	**PCI-X device**. For a detailed description, refer to the MindShare book entitled *PCI-X System Architecture* (published by Addison-Wesley).
08h	**Reserved for AMD**.
09h	**Vendor Specific capability register set**. The layout of the register set is vendor specific, except that the byte immediately following the "Next" pointer indicates the number of bytes in the capability structure (including the ID and Next pointer bytes). An example vendor specific usage is a function that is configured in the final manufacturing steps as either a 32-bit or 64-bit PCI agent and the Vendor Specific capability structure tells the device driver which features the device supports.
0Ah	**Debug port**.
0Bh	**CompactPCI central resource control**. A full definition of this capability can be found in the PICMG 2.13 Specification (http://www.picmg.com).

0Ch	**PCI Hot-Plug**. This ID indicates that the associated device conforms to the Standard Hot-Plug Controller model.
0Dh-0Fh	Reserved.
10h	**PCI Express Capability register set** (aka PCI Express Capability Structure). For a detailed explanation, refer to "PCI Express Capability Register Set" on page 896.
11h-FFh	Reserved.

Table 13: Currently-Assigned Capabilities IDs

CardBus CIS Pointer

This optional register is implemented by functions that share silicon between a CardBus device and a PCI or PCI Express function. This field points to the Card Information Structure (CIS) on the Card-Bus card. The register is read-only and indicates that the as can be accessed from the indicated offset within one of the following address spaces:

- Offset within the function's function-specific PCI-compatible configuration space (after DW 15d in the function's PCI-compatible configuration space).
- Offset from the start address indicated in one of the function's Memory Base Address Registers.
- Offset within a code image in the function's expansion ROM.

Expansion ROM Base Address Register

Required if a function incorporates a device ROM. Many PCI functions incorporate a device ROM (the spec refers to it as an expansion ROM) that contains a device driver for the function. The expansion ROM start memory address and size is specified in the Expansion ROM Base Address Register at configuration DW 12d in the configuration Header region. on power-up the system must be automatically configured so that each function's I0 and memory decoders recognize mutually-exclusive address ranges. The configuration software must be able to detect how much memory space an expansion ROM requires. In addition, the system must have the capability of programming a ROM's address decoder in order to locate its ROM in a non-conflicting address range.

When the start-up configuration program detects that a function has an Expansion ROM Base Address Register implemented (by writing all ones to it and reading it back), it must then check the first two locations in the ROM for an Expansion ROM signature to determine if a ROM is actually installed (i.e., there may be an empty ROM socket). If installed, the configuration program must shadow the ROM and execute its initialization code.

The format of the expansion ROM Base Address Register:

- A one in bit zero enables the function's ROM address decoder (assuming that the Memory Space bit in the Command register is also set to one).
- Bits [10:1] are reserved.

- Bits [31:11] are used to specify the ROM's start address (starting on an address divisible by the ROM's size).

As an example, assume that the programmer writes FFFFFFFEh to the ROM's Base Address Register (bit 0, the Expansion ROM Enable bit, is cleared so as not to enable the ROM address decoder until a start memory address has been assigned). A subsequent read from the register in the example yields FFFE0000h. This indicates the following:

- Bit 0 is a zero, indicating that the ROM address decoder is currently disabled.
- Bits [10:1] are reserved.
- In the Base Address field (bits [31:11]), bit 17 is the least-significant bit that the programmer was able to set to one. It has a binary-weighted value of 128K, indicating that the ROM decoder requires 128KB of memory space be assigned to the ROM. The programmer then writes a 32-bit start address into the register to assign the ROM start address on a 128K address boundary.

The PCI 2.3 spec recommends that the designer of the Expansion ROM Base Address Register should request a memory block slightly larger than that required by the current revision ROM to be installed. This permits the installation of subsequent ROM revisions that occupy more space without requiring a redesign of the logic associated with the function's Expansion ROM Base Address Register. The spec sets a limit of 16MB as the maximum expansion ROM size.

The Memory Space bit in the Command register has precedence over the Expansion ROM Enable bit. The function's expansion ROM should respond to memory accesses only if both its Memory Space bit (in its Command register) and the Expansion ROM Enable bit (in its expansion ROM Base Address register) are both set to one.

In order to minimize the number of address decoders that a function must implement, one address decoder can be shared between the Expansion ROM Base Address Register and one of the function's Memory Base Address Registers. The two Base Address Registers must be able to hold different values at the same time, but the address decoder will not decode ROM accesses unless the Expansion ROM Enable bit is set in the Expansion ROM Base Address Register.

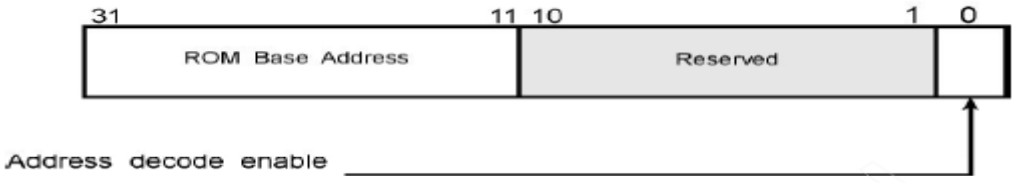


Figure 20: ROM Base Address Register

Command Register

16-Bit Register the software gives commands to the layer to do it as shown below. Some of the bits are read only as they do not apply for PCIe and some reserved and the most is read/Write.

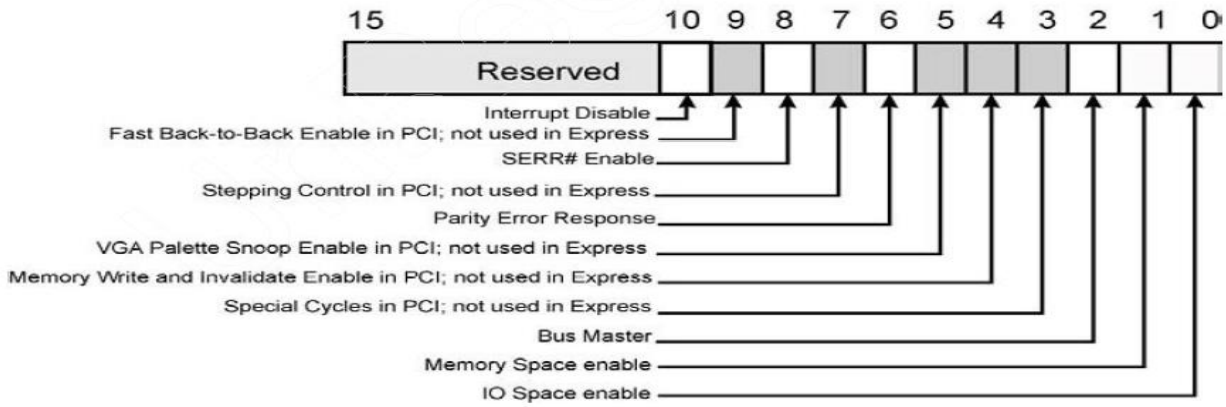


Figure 21: Command Register

Bit	Type	Description
0	RW	**IO Address Space Decoder Enable.** • **Endpoints:** – **0**. IO decoder is disabled and IO transactions targeting this device return completion with 'Unsupported Request' status. – **1**. IO decoder is enabled andIO transactions targeting this device are accepted.
1	RW	**Memory Address Space Decoder Enable.** • **Endpoints and Memory-mapped devices within Switch:** – **0**. Memory decoder is disabled and Memory transactions targeting this device return completion with 'Unsupported Request' completion status. – **1**. Memory decoder is enabled and Memory transactions targeting this device are accepted.
2	RW	**Bus Master Enable.** • **Endpoints:** – **0**. Disables an Endpoint function from issuing memory or IO requests. Also disables the ability to generate MSI messages. – **1**. Enables the Endpoint to issue memory or IO requests, including MSI messages. – Requests other than memory or IO requests are not controlled by this bit.

		– Default =0. – Hardwired to 0 if an Endpoint function does not generate memory or IO requests. • **Root and Switch Port**: – Controls the forwarding of memory or IO requests by a Switch or Root Port in the upstream direction. – **0**. Memory and IO requests received at a Root Port or the downstream side of a Switch port must return an Unsupported Requests (UR) Completion. – Does not affect the forwarding of Completions in either the upstream or downstream direction. – Does not control the forwarding of requests other than memory or IO requests. – Default value of this bit is 0b.
3	RO	**Special Cycle Enable**. Does not apply to PCI Express and must be 0.
4	RO	**Memory Write and Invalidate**. Does not apply to PCI Express and must be 0.
5	RO	**VGA Palette Snoop**. Does not apply to PCI Express and must be 0.
6	RW	**Parity Error Response**. In the Status register (see Figure 22-5 on page 780), the Master Data Parity Error bit is set by a Requester if its Parity Error Response bit is set and either of the following two conditions occurs: • If the Requester receives a poisoned Completion. • If the Requester poisons a write request. If the Parity Error Response bit is cleared, the Master Data Parity Error status bit is never set. The default value of this bit is 0.
7	RO	**IDSEL Stepping/Wait Cycle Control**. Does not apply to PCI Express and must be 0.
8	RW	**SERR Enable**. When set, this bit enables the non-fatal and fatal errors detected by the function to be reported to the Root Complex. The function reports such errors to the Root Complex if it is enabled to do so either through this bit or through the PCI Express specific bits in the Device Control register (see "Device Control Register" on page 905). The default value of this bit is 0.

9	RO	**Fast Back-to-Back Enable**. Does not apply to PCI Express and must be 0.
10	RW	**Interrupt Disable**. Controls the ability of a PCI Express function to generate INTx interrupt messages. • 0 = Function enabled to generate INTx interrupt messages. • 1 = Function's ability to generate INTx interrupt messages is disabled. If the function had already transmitted any Assert_INTx emulation interrupt messages and this bit is then set, it must transmit a corresponding Deassert_INTx message for each assert message transmitted earlier. Note that INTx emulation interrupt messages forwarded by Root and Switch Ports from devices downstream of the Root or Switch Port are not affected by this bit. The default value of this bit is 0.

Table 14: Command Register

Status Register

16-Bit Register the layer use it to update the software with status of what happened and happening in the layer. It has read only bits and read write one to clear bits. The read only is for bits which indicate something in the layer but RW1C bits indicates something has happened and the software write one to clear after reading it to enable the layer to detect it again like errors reporting.

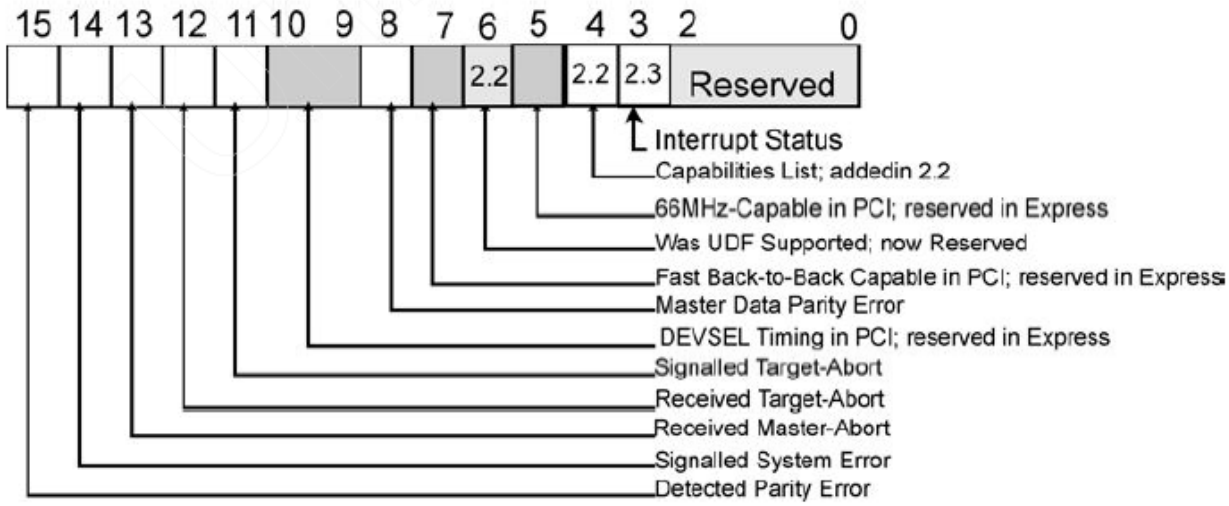


Figure 22: Status Register

Bit	Attributes	Description
3	RO	**Interrupt Status**. Indicates that the function has an interrupt request outstanding (that is, the function transmitted an interrupt message earlier in time and is awaiting servicing). Note that INTx emulation interrupts forwarded by Root and Switch Ports from devices downstream of the Root or Switch Port are not reflected in this bit. The default state of this bit is 0. *Note* : this bit is only associated with INTx messages, and has no meaning if the device is using Message Signaled Interrupts.
4	RO	**Capabilities List**. Indicates the presence of one or more extended capability register sets in the lower 48 dwords of the function's PCI-compatible configuration space. Since, at a minimum, all PCI Express functions are required to implement the PCI Express capability structure, this bit must be set to 1.
5	RO	**66MHz-Capable**. Does not apply to PCI Express and must be 0.
7	RO	**Fast Back-to-Back Capable**. Does not apply to PCI Express and must be 0.
8	RW1C	**Master Data Parity Error**. The Master Data Parity Error bit is set by a Requester if the Parity Error Enable bit is set in its Command register and either of the following two conditions occurs: • If the Requester receives a poisoned Completion. • If the Requester poisons a write request. If the Parity Error Enable bit is cleared, the Master Data Parity Error status bit is never set. The default value of this bit is 0.
10:9	RO	**DEVSEL Timing**. Does not apply to PCI Express and must be 0.

11	RW1C	**Signaled Target Abort**. This bit is set when a function acting as a Completer terminates a request by issuing Completer Abort Completion Status to the Requester. The default value of this bit is 0.
12	RW1C	**Received Target Abort**. This bit is set when a Requester receives a Completion with Completer Abort Completion Status. The default value of this bit is 0.
13	RW1C	**Received Master Abort**. This bit is set when a Requester receives a Completion with Unsupported Request Completion Status. The default value of this bit is 0.
14	RW1C	**Signaled System Error**. This bit is set when a function sends an ERR_FATAL or ERR_NONFATAL message, and the SERR Enable bit in the Command register is set to one. The default value of this bit is 0.
15	RW1C	**Detected Parity Error**. Regardless of the state the Parity Error Enable bit in the function's Command register, this bit is set if the function receives a Poisoned TLP. The default value of this bit is 0.

Table 15: Status Register

Interrupt line

Optional Register, A PCI Express function may generate interrupts in the legacy PCl/PCI-X manner. As an example, when a PCI Express-to-PCI or PCI-X bridge detects the assertion or deassertion of one of its INTA#, INTB#, INTC#, or INTD# inputs on the legacy side of the bridge, it sends an INTx Assert or Deassert message upstream towards the Root Complex (specifically, to the interrupt controller within the Root Complex).

As in PCI, the Interrupt Line register communicates interrupt line routing information. The register is read/write and must be implemented by any function that contains a valid non-zero value in its Interrupt Pin configuration register. The OS or device driver can examine a device's Interrupt Line register to determine which system interrupt request line the device uses to issue requests for service (and, therefore, which entry in the interrupt table to "hook").

In a non-PC environment, the value written to this register is architecture-specific and therefore outside the scope of the specification.

Interrupt Pin

This read-only optional register identifies the legacy INTx interrupt Message (INTA, INTB, INTC, or INTD) the function transmits upstream to generate an interrupt. The values Olh-through-04h correspond to legacy INTx interrupt Messages INTA-through-INTD. A

return value of zero indicates that the device doesn't generate interrupts using the legacy method. All other values (05h-FFh) are reserved. Note that, although the function may not generate interrupts via the legacy method, it may generate them via the MSI method.

Base Address Registers (BARs)

On power-up, the system must be automatically configured so that each function's I0 and memory functions occupy mutually-exclusive address ranges. In order to accomplish this, the system must be able to detect how many memory and 10 address ranges a function requires and the size of each. Obviously, the system must then be able to program the function's address decoders in order to assign non-conflicting address ranges to them.

The Base Address Registers (BARs), located in DWs 4-through-9 of the function's configuration Header space, are used to implement a function's programmable memory and/or I0 decoders. Each register is 32-bits wide (or 64-bits wide if it's a memory decoder and its associated memory block can be located above the 4GB address boundary). the three possible formats of a Base Address Register are 32-bit & 64-bit Memory & 32-bit IO. Bit 0 is a read-only bit and indicates whether it's a memory or an I0 decoder:

- If bit 0 = 0, the register is a memory address decoder.
- If bit 0 = 1, the register is an I0 address decoder.

Decoders may be implemented in any of the Base Address Register positions. If more than one decoder is implemented, there may be holes. During configuration, the configuration software must therefore look at all six of the possible Base Address Register positions in a function's Header to determine which registers are actually implemented.

In Memory BARs in Third bit is Prefetchable Attribute Bit. Bit three defines the block of memory as Prefetchable or not. A block of memory space may be marked as Prefetchable only if it can guarantee that:

- there are no side effects from reads (e.g., the read doesn't alter the con-tents of the location or alter the state of the function in some manner). It's permissible for a bridge that resides between a Requester and a memory target to prefetch read data from memory that has this characteristic. If the Requester doesn't end up asking for all of the data that the bridge read into a read-ahead buffer, the bridge must discard the data. The data remains unchanged in the target's memory locations.
- on a read, it always returns all bytes irrespective of the byte enable set-tings.
- the memory device continues to function correctly if a bridge that resides between the Requester and the memory target performs byte merging in its posted memory write buffer when memory writes are performed within the memory target's range.

The device uses two BARs in case of addressed 32-bit memory addressing one for start and one for end & in case of addressed 64-bit memory addressing uses 4 BARs 2 for start and 2 for end & in case of IO 32-bit Addressing uses 2 BARs for start and end

Worst Case is when device is addressed by IO & 64-bit Memory which uses all the 6 BARs.

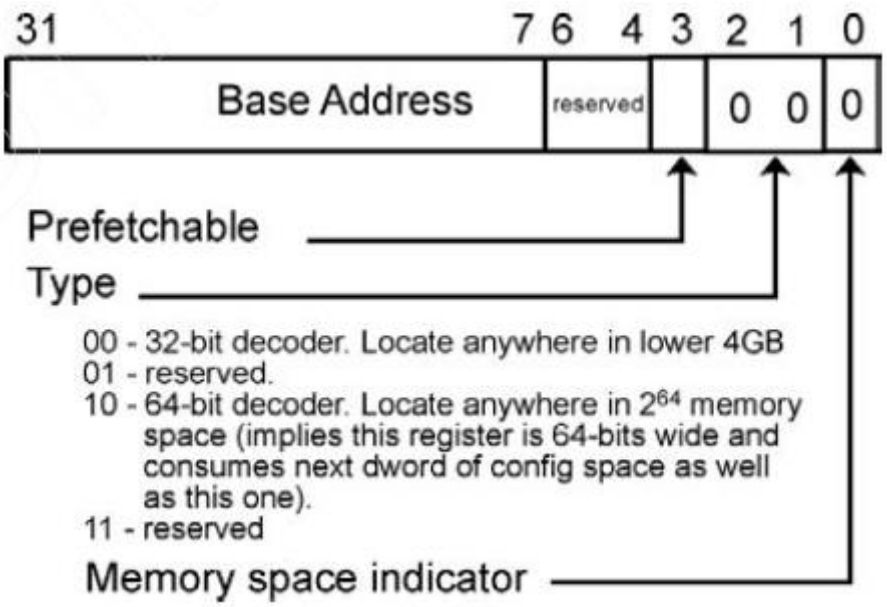


Figure 23: BAR 32-bit Memory assignment

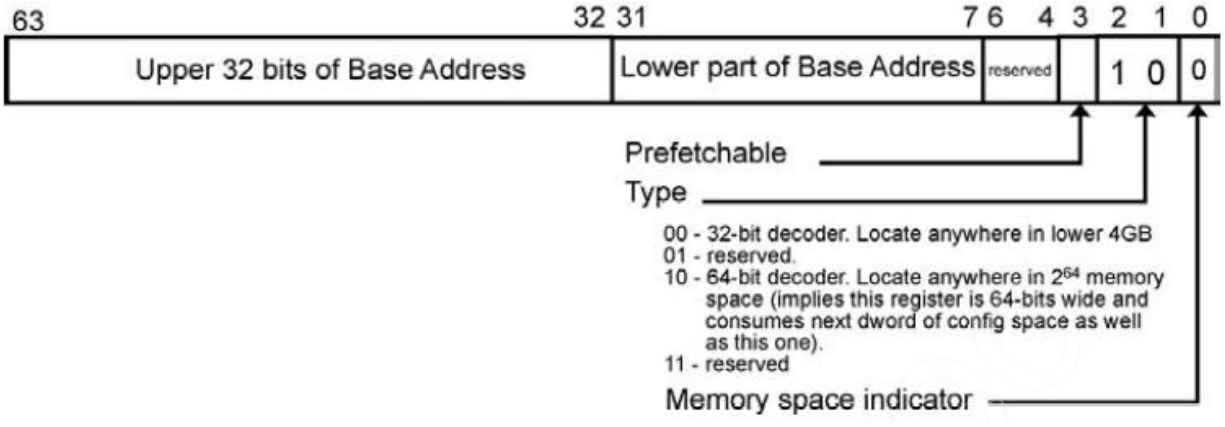


Figure 24: BAR 64-bit Memory assignment

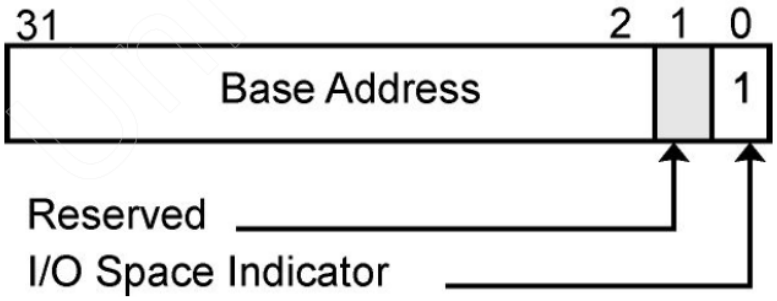


Figure 25: IO bit Assignment

A Memory Example. As an example, assume that FFFFFFFFh is written to the Base Address Register at configuration DW 04d and the value read back is FFF00000h. The fact that any bits could be changed to one indicates that the Base Address Register is implemented.

- Bit 0 = 0, indicating that this is a memory address decoder.
- Bits [2:1] = 00b, indicating that it's a 32-bit memory decoder.

- Bit 3 = 0, indicating that it's not Prefetchable memory
- Bit 20 is the first one bit found in the Base Address field. The binary-weighted value of this bit is 1,048,576, indicating that this is an address decoder for 1MB of memory.

The programmer then writes a 32-bit base address into the register. How-ever, only bits [31:20] are writable. The decoder accepts bits [31:20] and assumes that bits [19:0] of the assigned base address are zero. This means that the base address is divisible by 1MB, the size of the requested memory range. It is a characteristic of PCI, PCI-X, and PCI Express decoders that the assigned start address is always divisible by the size of requested range.

As an example, it is possible to program the example memory address decoder for a 1MB block of memory to start on the one, two, or three meg boundary, but it is not possible to set its start address at the 1.5, 2.3, or 3.7 meg boundary.

An IO Example. As a second example, assume that FFFFFFFFh is written to a function's Base Address Register at configuration DW address 05d and the value read back is FFFFFF01h. Bit 0 is a one, indicating that this is an IO address decoder. Scanning upwards starting at bit 2 (the least-significant bit of the Base Address field), bit 8 is the first bit that was successfully changed to one. The binary-weighted value of this bit is 256, indicating that this is an JO address decoder requesting 256 bytes of IO space.

The programmer then writes a 32-bit base JO address into the register. However, only bits [31:8] are writable. The decoder accepts bits [31:8] and assumes that bits [7:0] of the assigned base address are zero. This means that the base address is divisible by 256, the size of the requested IO range.

3-5-3 Type 1

Bus Numbers Registers

Before proceeding, it's important to define some basic terms associated with an actual or a virtual PCI-to-PCI bridge. Each PCI-to-PCI bridge is connected to two buses, referred to as its primary and secondary buses:

- Downstream: When a transaction is initiated and is passed through one or more PCI-to-PCI bridges flowing away from the host processor, it is said to be moving downstream.
- Upstream: When a transaction is initiated and is passed through one or more PCI-to-PCI bridges flowing towards the host processor, it is said to be moving upstream.
- Primary bus: PCI bus that is directly connected to the upstream side of a bridge.
- Secondary bus: PCI bus that is directly connected to the downstream inter-face of a PCI-to-PCI bridge.
- Subordinate bus: Highest-numbered PCI bus on the downstream side of the bridge.

Introduction

Each PCI-to-PCI bridge must implement three mandatory bus number registers. All of them are read /writable and are cleared to zero by reset. During configuration, the

configuration software initializes these three registers to assign bus numbers. These registers are:

- the Primary Bus Number register.
- the Secondary Bus Number register.
- the Subordinate Bus Number register.

The combination of the Secondary and the Subordinate Bus Number register values defines the range of buses that exists on the downstream side of the bridge. The information supplied by these three registers is used by the bridge to determine whether or not to pass a packet through to the opposite interface.

Primary bus register

PCI-Compatible register. Mandatory. Located in Header byte zero of DW six. The Primary Bus Number register is initialized by software with the number of the bus that is directly connected to the bridge's primary interface. This register exists for three reasons:

- To mute Completion packets.
- To mute a Vendor-defined message that uses ID-based routing.
- To route a PCI Special Cycle Request that is moving upstream.

Secondary bus register

PCI-Compatible register. Mandatory. Located in Header byte one of DW six. The Secondary Bus Number register is initialized by software with the number of the bus that is directly connected to the bridge's secondary interface. This register exists for three reasons:

- When a Special Cycle Request is latched on the primary side, the bridge uses this register (and, possibly, the Subordinate Bus Number register) to determine if it should be passed to the bridge's secondary interface as either a PCI Special Cycle transaction (if the bus connected to the secondary inter-face is the destination PCI or PCI-X bus) or as is (i.e., as a Type 1 configuration write request packet).
- When a Type 1 Configuration transaction (read or write and not a PCI Spe-cial Cycle Request) is latched on the primary side, the bridge uses this regis-ter (and, possibly, the Subordinate Bus Number register) to determine if it should be passed to the bridge's secondary interface as either a Type 0 con-figuration transaction (if the bus connected to the secondary interface is the destination PCI or PCI-X bus) or as is (i.e., as a Type 1 configuration write request packet).
- When a Completion packet is latched on the primary side, the bridge uses this register (and, possibly, the Subordinate Bus Number register) to deter-mine if it should be passed to the bridge's secondary interface.

Subordinate bus register

PCI-Compatible register. Mandatory. Located in Header byte two of DW six. The Subordinate Bus Number register is initialized by software with the number of the highest-numbered bus that exists on the downstream side of the bridge. If there are no PCI-to-PCI

bridges on the secondary bus, the Subordinate Bus Number register is initialized with the same value as the Secondary Bus Number register.

Bridge Routes ID Addressed Packets Using Bus Number Registers

When one of the bridge's interfaces latches a Completion packet, an ID-routed Vendor-defined message, or a PCI Special Cycle request, it uses its internal bus number registers to decide whether or not to accept the packet and pass it to the opposite bridge interface:

- The routing of PCI Special Cycle requests was described in the previous sections.
- When the bridge latches a Completion packet or an ID-routed Vendor-defined message on its primary interface, it compares the Bus Number portion of destination ID to its Secondary Bus Number and Subordinate Bus Number register values. If the target bus number falls within the range of buses defined by the bridge's Secondary Bus Number and Subordinate Bus Number registers, the bridge accepts the packet and passes it to its opposite interface.
- When the bridge latches a Completion packet or an ID-routed Vendor-defined message on its secondary interface, it compares the Bus Number portion of the destination ID to its Primary Bus Number register.
    - If it matches, the bridge accepts the packet and passes it to the primary interface.
    - If it doesn't match the bridge's Primary Bus Number register and it's outside the range of buses defined by the bridge's Secondary Bus Number and Subordinate Bus Number registers, the bridge accepts the packet (the target bus is not on the downstream side of the bridge and therefore it must be passed upstream) and passes it to its primary inter-face.
    - If the destination bus falls within the range of buses defined by the bridge's Secondary Bus Number and Subordinate Bus Number registers, then the target bus is on the downstream side of the bridge. The bridge therefore does not accept the packet.
- These registers are also used to route Type 1 configuration packets.

Base Address Registers

Differs from PCI. Optional. Only necessary if the bridge implements a device-specific register set and/or a memory buffer. Located in Header DWs four and five. If the designer doesn't implement any internal, device-specific register set or memory, then these address decoders aren't necessary. These Base Address Registers are used in the same manner as those described for a non-bridge PCI function. If implemented, both may be implemented as memory decoders, both as I0 decoders, one as memory and one as I0, or only one may be implemented as either I0 or memory.

If a BAR is implemented as a memory BAR with the Prefetchable bit set to one, it must be implemented as a 64-bit memory BAR and would therefore consume both DWs four and five.

IO & Memory & Prefetchable Memory Base and Limit Registers

Used to know if the incoming packet is routed to an endpoint or bridge on the downstream of this bridge or not. It can route 3 types:

- IO 16&32-bit (In case of 32-bit it uses the upper IO registers to complete the address)
- Memory 32-bit
- Prefetchable 32&64-bit (In case of 64-bit it uses the upper IO registers to complete the address)

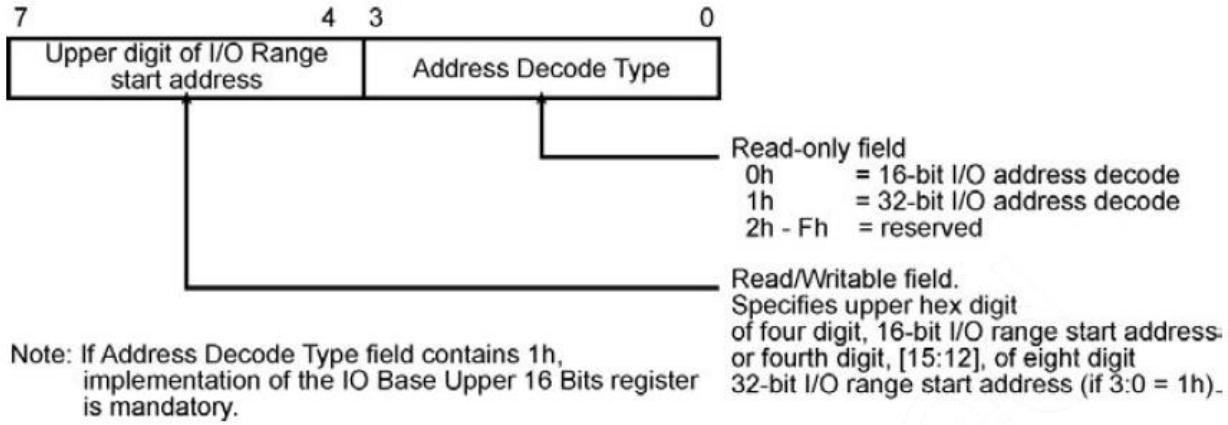


Figure 26: IO Base & Limit Registers bit assignment

IO 16-bit Example. Assume that the I0 base is set to 2h and the I0 Limit is set to 3h. The bridge is now primed to recognize any I0 transaction on the primary bus that targets an I0 address within the range consisting of 2000h through 3FFFh.

Anytime that the bridge detects an I0 transaction on the primary bus with an address inside the 2000h through 3FFFh range, it accepts the transaction and passes it through (because it's within the range defined by the I0 Base and Limit registers and may therefore be for an I0 device that resides behind the bridge).

Anytime that the bridge detects an IO transaction on the primary bus with an address outside the 2000h through 3FFFh range, it ignores the transaction (because the target 10 address is outside the range of addresses assigned to I0 devices that reside behind the bridge).

Anytime that the bridge detects an I0 transaction on the secondary bus with an address inside the 2000h through 3FFFh range, it ignores the transaction (because the target address falls within the range assigned to I0 devices that reside on the secondary side of the bridge).

Anytime that the bridge detects an I0 transaction on the secondary bus with an address outside the 2000h through 3FFbh range, it accepts the transaction and passes it through to the primary side (because the target address falls outside the range assigned to 10 devices that reside on the secondary side of the bridge, but it may be for an I0 device on the primary side).

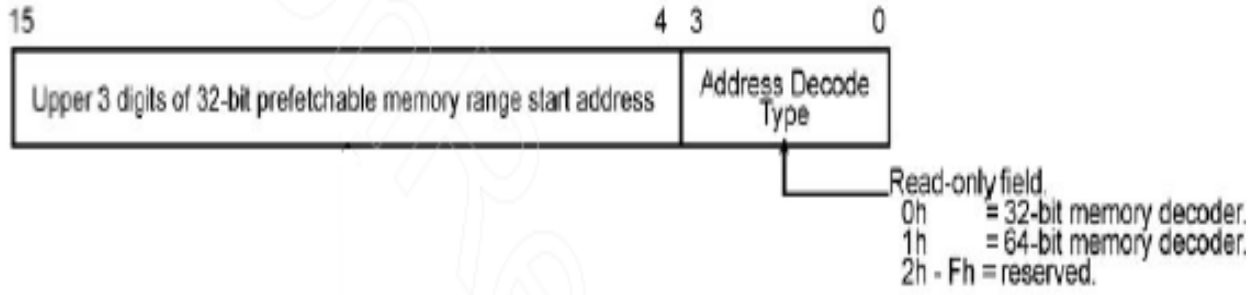


Figure 27: Prefetchable memory Base & Limit Registers

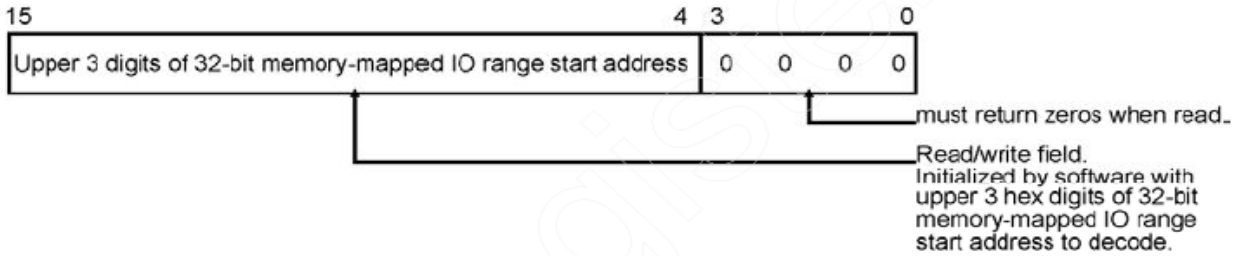


Figure 28: memory Base & Limit Registers

As Prefetchable 64-bit example, assume that these four registers are set as follows:

- FF00000h is written into the Prefetchable Memory Base Upper 32-bits register.
- 123h is written into the upper three digits of the Base register.
- FF000000h is written into Prefetchable Memory Limit Upper 32-bits register.
- 124h is written into the upper three digits of the Limit register.

This defines the Prefetchable memory address range as the 2MB range from FF00000012300000h through FF000000124FFFFFh.

As another example, assume they are programmed as follows:

- 00000230h is written into the Prefetchable Memory Base Upper 32-bits register.
- 222h written into the upper three digits of the Base register.
- 00000230h is written into Prefetchable Memory Limit Upper 32-bits register.
- 222h written into the upper three digits of the Limit register.

This defines the Prefetchable memory address range as the 1MB range from 0000023022200000h through 00000230222FFFFFh.

As a memory 32-bit example, assume that the configuration software has written the following values to the Memory Base and Limit registers:

- The upper three digits of the Memory Base register contain 555h.
- The upper three digits of the Memory Limit register contain 678h.

This defines a 292MB memory-mapped I0 region on the downstream side of the bridge starting at 55500000h and ending at 678FEFFFh.

Command & Control Registers

The bridge designer must implement two required command registers in the bridge's configuration Header region:

- The Command register is the standard configuration Command register defined by the spec for any function. It is associated with the bridge's primary bus interface.
- The Bridge Control register is an extension to the standard Command register and is associated with the operation of both of the bridge's bus interfaces.

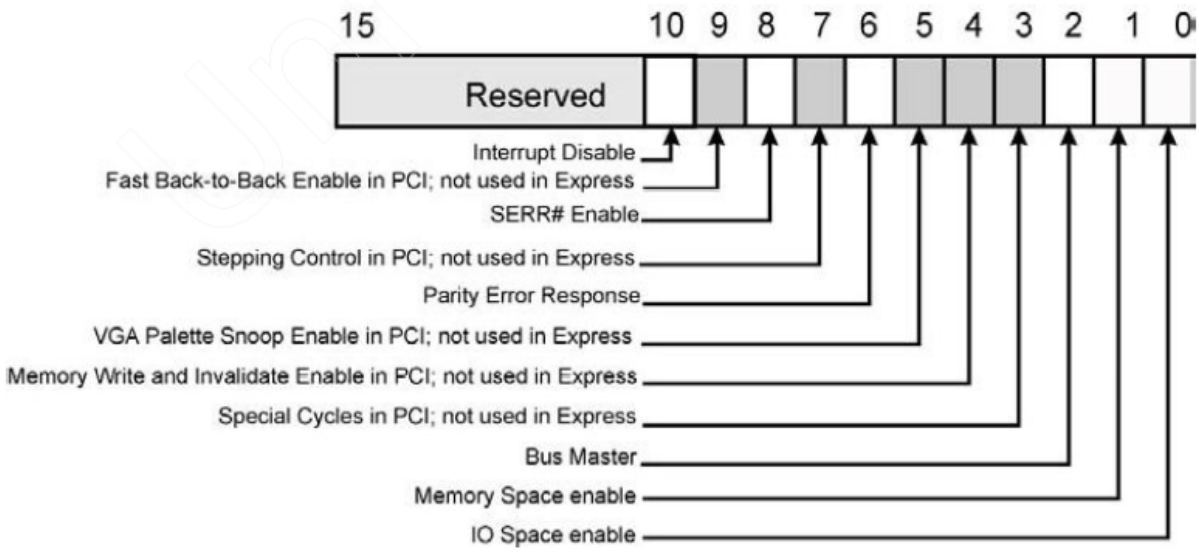


Figure 29: Bridge Command Register

Bit	Attributes	Description
0	RW	**IO Address Space Decoder Enable.** – **0**. IO transactions received at the downstream side of a bridge that are moving in the upstream direction are not forwarded and bridge returns completion with 'Unsupported Request' completion status. – **1**. IO transactions received at the downstream side of a bridge that are moving in the upstream direction are forwarded from the secondary to primary side of the bridge.
1	RW	**Memory Address Space Decoder Enable.** • **Memory-mapped devices within Bridge**: – **0**. Memory decoder is disabled and Memory transactions targeting this device results in bridge returning completion with 'Unsupported Request' completion status. – **1**. Memory decoder is enabled and Memory transactions targeting this device are accepted. • **Memory transactions targeting device on the upstream side of a bridge**: – **0**. Memory transactions received at the downstream side of a bridge are not forwarded to the upstream side and bridge returns 'Unsupported Request' completion status. – **1**. Memory transactions received at the dowstream side of a bridge that target a device residing on the upstream side of a bridge are forwarded from the secondary to the primary side of the bridge.
2	RW	**Bus Master.** – Controls the ability of a Root Port or a downstream Switch Port to forward memory or IO requests in the upstream direction. – **If this bit is 0,** when a Root Port or a downstream Switch Port receives an upstream-bound memory request or IO request, it returns Unsupported Requests (UR) status to the requester. – This bit does not affect forwarding of Completions in either the upstream or downstream direction. – The forwarding of requests other than those mentioned above are not controlled by this bit. – Default value of this bit is 0.

3	RO	**Special Cycles**. Does not apply to PCI Express and must be hard-wired to 0.
4	RO	**Memory Write and Invalidate Enable**. Does not apply to PCI Express and must be hardwired to 0.
5	RO	**VGA Palette Snoop**. Does not apply to PCI Express and must be hardwired to 0.
6	RW	**Parity Error Response**. When forwarding a Poisoned TLP from Primary to Secondary: • The primary side must set the Detected Parity Error bit in the bridge Status register. • If the Parity Error Response bit in the Bridge Control register is set, the secondary side must set the Master Data Parity Error bit in the Secondary Status register. When forwarding a Poisoned TLP from Secondary to Primary: • The secondary side must set the Detected Parity Error bit in the Secondary Status register. • If the Parity Error Response bit in the Bridge Control register is set, the primary side must set the Master Data Parity Error bit in the bridge Status register. If the Parity Error Response bit is cleared, the Master Data Parity Error status bit in the bridge Status register is never set. The default value of this bit is 0.
7	RO	**Stepping Control**. Does not apply to PCI Express. Must be hard-wired to 0.
8	RW	**SERR# Enable**. When set, this bit enables the non-fatal and fatal errors detected by the bridge's primary interface to be reported to the Root Complex. The function reports such errors to the Root Complex if it is enabled to do so either through this bit or through the PCI Express specific bits in the Device Control register (see "Device Control Register" on page 905). The default value of this bit is 0.
9	RO	**Fast Back-to-Back Enable.** Does not apply to PCI Express and must be hardwired to 0.

10	RW	**Interrupt Disable**. Controls the ability of a bridge to generate INTx interrupt messages: • 0 = The bridge is enabled to generate INTx interrupt messages. • 1 = The bridge's ability to generate INTx interrupt messages is disabled. If the bridge had already transmitted any Assert_INTx emulation interrupt messages and this bit is then set, it must transmit a corresponding Deassert_INTx message for each assert message transmitted earlier. Note that INTx emulation interrupt messages forwarded by Root and Switch Ports from devices downstream of the Root or Switch Port are not affected by this bit. The default value of this bit is 0.
15:11		**Reserved**. Read-only and must return zero when read.

Table 26: Bridge Command Register

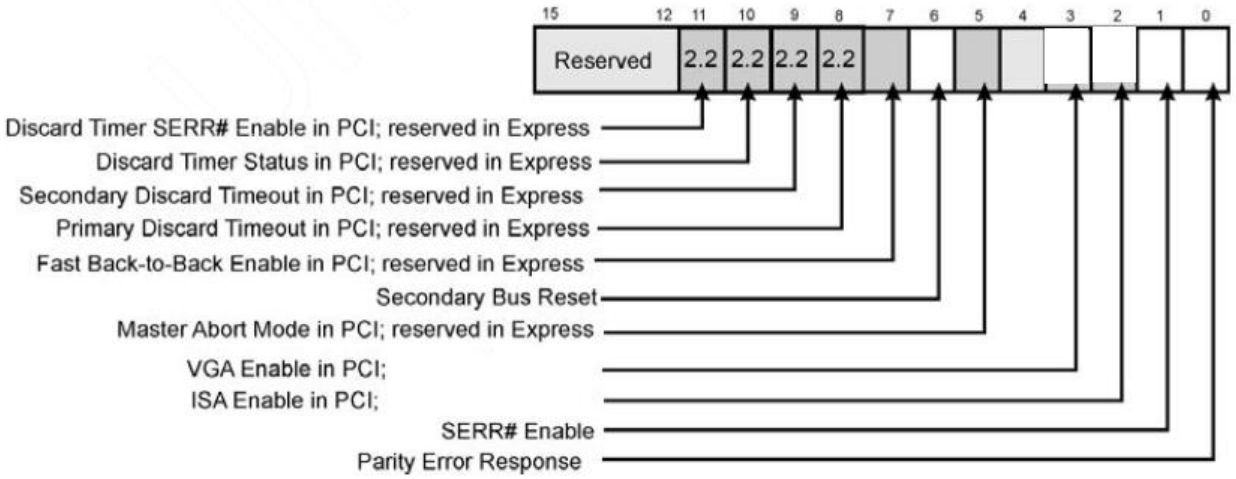


Figure 30: Bridge Control Register

Bit	Attributes	Description
0	RW	**Parity Error Response**. When forwarding a Poisoned TLP from Primary to Secondary: • The primary side must set the Detected Parity Error bit in the bridge Status register. • If the Parity Error Response bit in the Bridge Control register is set, the secondary side must set the Master Data Parity Error bit in the Secondary Status register. When forwarding a Poisoned TLP from Secondary to Primary: • The secondary side must set the Detected Parity Error bit in the Secondary Status register. • If the Parity Error Response bit in the Bridge Control register is set, the primary side must set the Master Data Parity Error bit in the bridge Status register. If the Parity Error Response bit is cleared, the Master Data Parity Error status bit in the Secondary Status register is never set. The default value of this bit is 0.
1	RW	**SERR# Enable.** This bit controls the forwarding of ERR_COR (correctable errors), ERR_NONFATAL (non-fatal errors), and ERR_FATAL (fatal errors) received on the secondary side to the primary side. Default value of this field is 0.
2	RW	**ISA Enable.** See page 582 of the MindShare PCI book.
3	RW	**VGA Enable.** See page 608 of the MindShare PCI book.
4	RO	**Reserved.** Hardwired to zero.
5	RO	**Master Abort Mode.** Not used in Express and must be hardwired to zero.
6	RW	**Secondary Bus Reset.** Setting this bit to one triggers a hot reset on the Express downstream port. Port configuration registers must not be affected, except as required to update port status. Default value of this field is 0.
7	RO	**Fast Back-to-Back Enable.** Not used in Express and must be hardwired to zero.

8	RO	**Primary Discard Timeout**. Not used in Express and must be hard-wired to zero.
9	RO	**Secondary Discard Timeout**. Not used in Express and must be hardwired to zero.
10	RO	**Discard Timer Status**. Not used in Express and must be hard-wired to zero.
11	RO	**Discard Timer SERR# Enable**. Not used in Express and must be hardwired to zero.

Table 17: Bridge Control Register

Status & Secondary Status Register

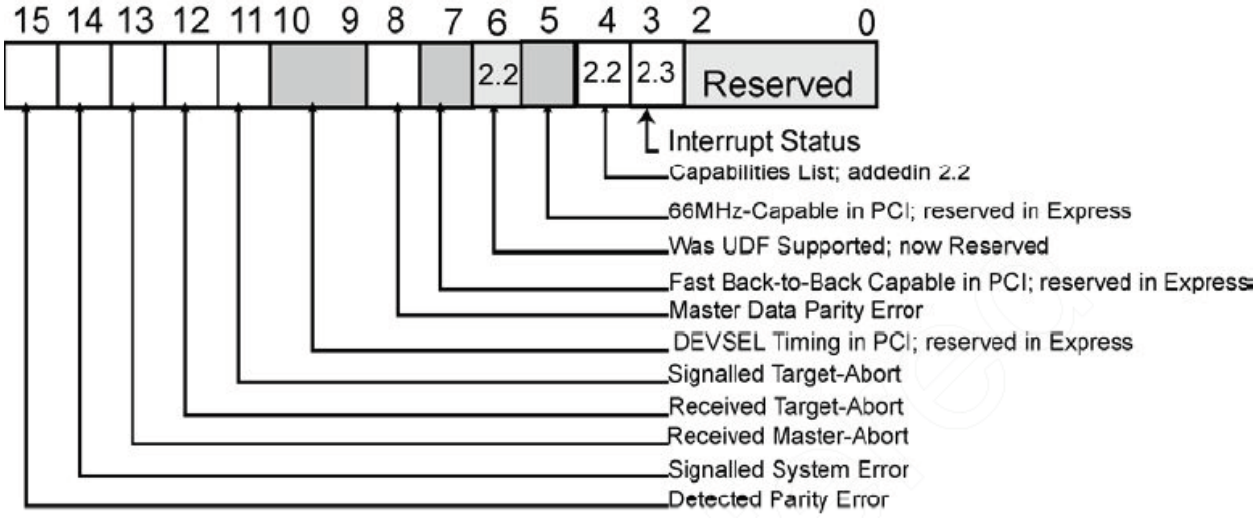


Figure 31: Bridge Status Register

Bit	Attributes	Description
3	RO	**Interrupt Status**. Indicates that the bridge itself had previously transmitted an interrupt request to its driver (that is, the function transmitted an interrupt message earlier in time and is awaiting servicing). Note that INTx emulation interrupts forwarded by Root and Switch Ports from devices downstream of the Root or Switch Port are not reflected in this bit. The default state of this bit is 0.
4	RO	**Capabilities List**. Indicates the presence of one or more extended capability register sets in the lower 48 dwords of the function's PCI-compatible configuration space. Since, at a minimum, all PCI Express functions are required to implement the PCI Express capability structure, this bit must be set to 1.
5	RO	**66MHz-Capable**. Does not apply to PCI Express and must be 0.
7	RO	**Fast Back-to-Back Capable**. Does not apply to PCI Express and must be 0.
8	RW1C	**Master Data Parity Error**. When forwarding a Poisoned TLP from Primary to Secondary: • The primary side must set the Detected Parity Error bit in the bridge Status register. • If the Parity Error Response bit in the Bridge Control register is set, the secondary side must set the Master Data Parity Error bit in the Secondary Status register. When forwarding a Poisoned TLP from Secondary to Primary: • The secondary side must set the Detected Parity Error bit in the Secondary Status register. • If the Parity Error Response bit in the Bridge Control register is set, the primary side must set the Master Data Parity Error bit in the bridge Status register. If the Parity Error Response bit in the Bridge Command register is cleared, the Master Data Parity Error status bit in the Secondary Status register is never set. The default value of this bit is 0.
10:9	RO	**DEVSEL Timing**. Does not apply to PCI Express and must be 0.

11	RW1C	**Signaled Target Abort**. This bit is set when the bridge's primary interface completes a received request by issuing a Completer Abort Completion Status. Default value of this field is 0.
12	RW1C	**Received Target Abort**. This bit is set when the bridge's primary interface receives a Completion with Completer Abort Completion Status. Default value of this field is 0.
13	RW1C	**Received Master Abort**. This bit is set when the bridge's primary interface receives a Completion with Unsupported Request Completion Status. Default value of this field is 0.
14	RW1C	**Signaled System Error**. This bit is set when the bridge's primary interface sends an ERR_FATAL (fatal error) or ERR_NONFATAL (non-fatal error) message (if the SERR Enable bit in the bridge Command register is set to one). The default value of this bit is 0.
15	RW1C	**Detected Parity Error**. This bit is set by the bridge's primary interface whenever it receives a Poisoned TLP, regardless of the state the Parity Error Enable bit in the bridge Command register. Default value of this bit is 0.

Table 18: Bridge Status Register

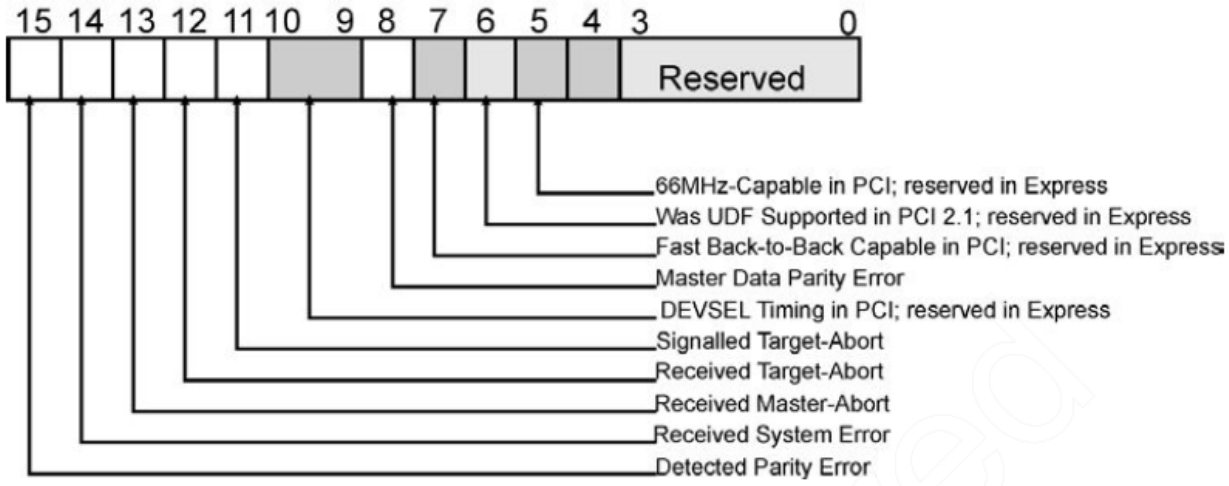


Figure 32: Bridge Secondary Status Register

Bit	Attributes	Description
5	RO	**66MHz-Capable**. Does not apply to Express and must be 0.
7	RO	**Fast Back-to-Back Capable**. Does not apply to PCI Express and must be 0.
8	RW1C	**Master Data Parity Error**. When forwarding a Poisoned TLP from Primary to Secondary: • The primary side must set the Detected Parity Error bit in the bridge Status register. • If the Parity Error Response bit in the Bridge Control register is set, the secondary side must set the Master Data Parity Error bit in the Secondary Status register. When forwarding a Poisoned TLP from Secondary to Primary: • The secondary side must set the Detected Parity Error bit in the Secondary Status register. • If the Parity Error Response bit in the Bridge Control register is set, the primary side must set the Master Data Parity Error bit in the bridge Status register. If the Parity Error Response bit in the Bridge Control register is cleared, the Master Data Parity Error status bit in the Secondary Status register is never set. The default value of this bit is 0.
10:9	RO	**DEVSEL Timing**. Does not apply to Express and must be 0.
11	RW1C	**Signaled Target Abort**. This bit is set when the bridge's secondary interface completes a received request by issuing a Completer Abort Completion Status. Default value of this field is 0.

12	RW1C	**Received Target Abort**. This bit is set when the bridge's secondary interface receives a Completion with Completer Abort Completion Status. Default value of this field is 0.
13	RW1C	**Received Master Abort**. This bit is set when the bridge's secondary interface receives a Completion with Unsupported Request Completion Status. Default value of this field is 0.
14	RW1C	**Signaled System Error**. This bit is set when the bridge's secondary interface sends an ERR_FATAL (fatal error) or ERR_NONFATAL (non-fatal error) message (if the SERR Enable bit in the Bridge Control register is set to one. The default value of this bit is 0.
15	RW1C	**Detected Parity Error**. This bit is set by the bridge's secondary interface whenever it receives a Poisoned TLP, regardless of the state the Parity Error Enable bit in the Bridge Control register. Default value of this bit is 0.

Table 19 Bridge Secondary Status Register

3-5-4 Capabilities

PCIe Capability

This capability gives more control to the software to the device, the capability register is read only register to tell the software about what is the device hardware capable of, the control is like command register to tell the device what to use of this hardware.

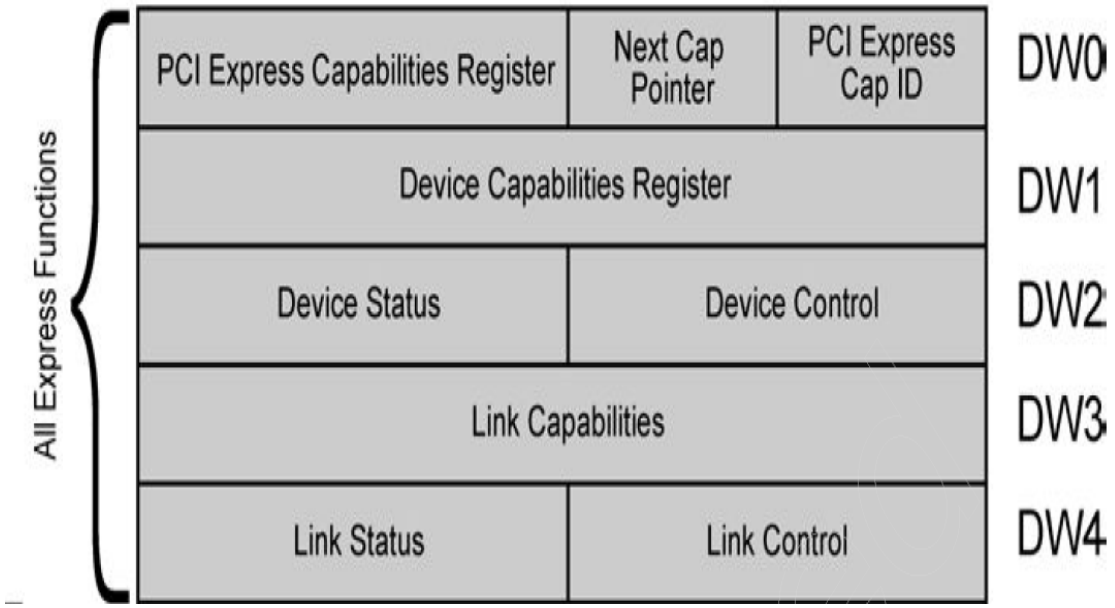


Figure 33: PCIe Capability Registers

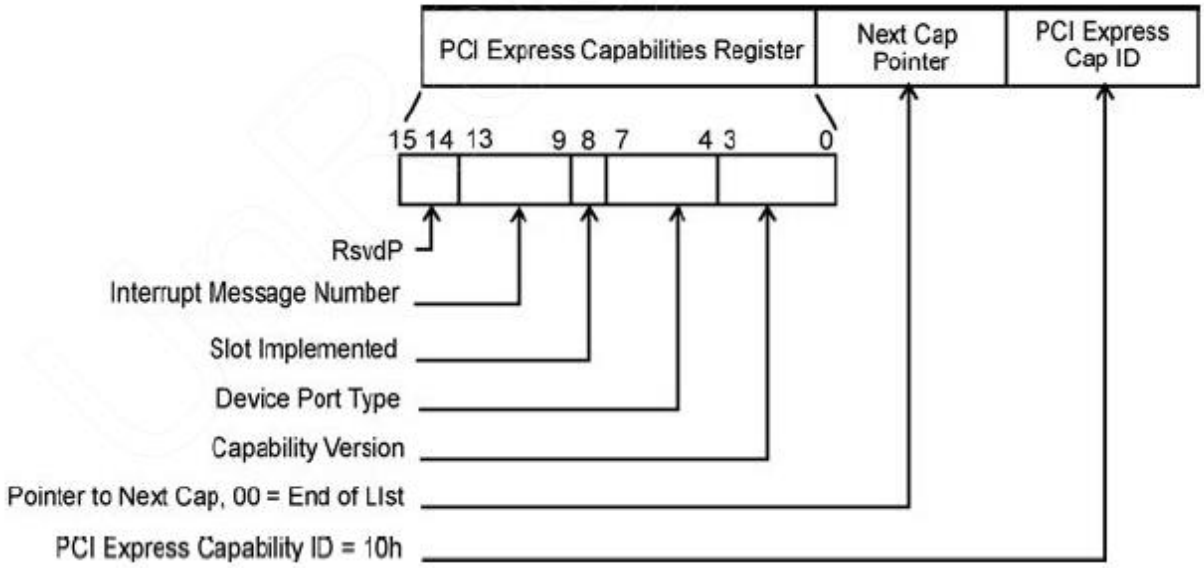


Figure 34: PCIe Capability Register

Bit(s)	Type	Description
3:0	RO	**Capability Version**. SIG-defined PCI Express capability structure version number (must be 1h).
7:4	RO	**Device/Port Type**. Express logical device type: • 0000b: **PCI Express Endpoint**. Some OSs and/or processors may not support IO accesses (i.e., accesses using IO rather than memory addresses). This being the case, the designer of a native PCI Express function should avoid the use of IO BARs. However, the target system that a function is designed for may use the function as one of the boot devices (i.e., the boot input device (e.g., keyboard), output display device, or boot mass storage device) and may utilize a legacy device driver for the function at startup time. The legacy driver may assume that the function's device-specific register set resides in IO space. In this case, the function designer would supply an IO BAR to which the configuration software will assign an IO address range. When the OS boot has completed and the OS has loaded a native PCI Express driver for the function, however, the OS may deallocate all legacy IO address ranges previously assigned to the selected boot devices. From that point forward and for the duration of the power-up session, the native driver will utilize memory accesses to communicate with its associated function through the function's memory BARs. • 0001b: **Legacy PCI Express Endpoint**. A function that requires IO space assignment through BARs for run-time operations. Extended configuration space capabilities, if implemented on legacy PCI Express Endpoint devices, may be ignored by software. • 0100b: **Root Port** of PCI Express Root Complex*. • 0101b: **Switch upstream port***. • 0110b: **Switch downstream port***. • 0111b: **Express-to-PCI/PCI-X bridge***. • 1000b: **PCI/PCI-X to Express bridge***. • All other encodings are reserved. * Only valid for functions with a Type 1 configuration register layout.

8	HWInit	**Slot Implemented**. When set, indicates that this Root Port or Switch downstream port is connected to an add-in card slot (rather than to an integrated component or being disabled). See "Chassis and Slot Number Assignment" on page 861 for more information.
13:9	RO	**Interrupt Message Number**. If this function is allocated more than one MSI interrupt message value (see "Message Data Register" on page 335), this register contains the MSI Data value that is written to the MSI destination address when any status bit in either the Slot Status register (see "Slot Status Register" on page 925) or the Root Status register (see "Root Status Register" on page 928) of this function are set. If system software should alter the number of message data values assigned to the function, the function's hardware must update this field to reflect the change.

Table 20: PCIe Capability Register

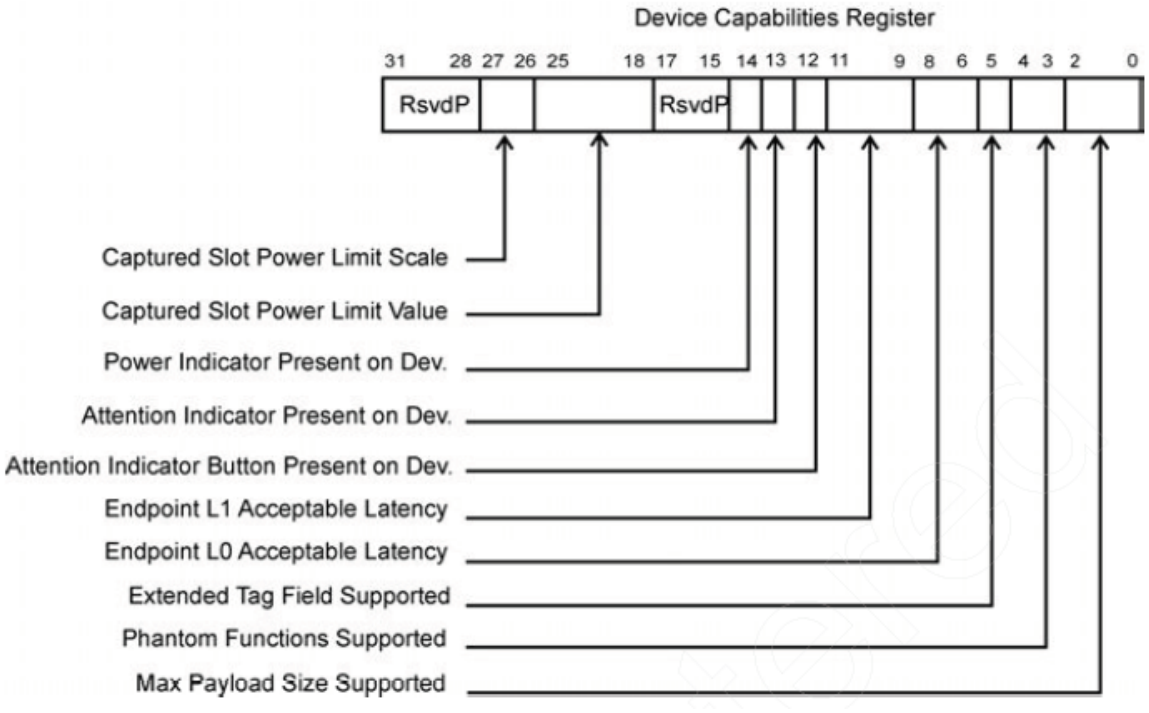


Figure 35: Device Capabilities Register

Bit(s)	Description
2:0	**Max Payload Size Supported**. Max data payload size that the function supports for TLPs: • 000b = 128 bytes max payload size • 001b = 256 bytes max payload size • 010b = 512 bytes max payload size • 011b = 1KB max payload size • 100b = 2KB max payload size • 101b = 4KB max payload size • 110b = Reserved • 111b = Reserved
4:3	**Phantom Functions Supported**. **Background**: Normally, each Express function (when acting as a Requester) is limited to no more than 32 outstanding requests currently awaiting completion (as indicated by the lower five bits of the transaction Tag; the upper three bits of the Tag must be zero). However, a function may require more than this. If the Extended Tag Field is supported (see bit 5 in this table) and if the Extended Tag Field Enable bit in the Device Control register is set (see "Device Control Register" on page 905), the max is increased to 256 and all eight bits of the Requester ID Tag field are used when a function within the device issues a request packet. If a function requires a greater limit than 256, it may do this via Phantom Functions. **Description**: When the device within which a function resides does not implement all eight functions, a non-zero value in this field indicates that this is so. Assuming all functions are not implemented and that the programmer has set the Phantom Function Enable bit in the Device Control register (see "Device Control Register" on page 905), a function may issue request packets using its own function number as well as one or more additional function numbers. This field indicates the number of msbs of the function number portion of Requester ID that are logically combined with the Tag identifier. • 00b. The **Phantom Function feature is not available** within this device. • 01b. The msb of the function number in the Requestor ID is used for Phantom Functions. The device designer may implement functions 0-3. When issuing request packets, **Functions 0, 1, 2, and 3 may also use function numbers 4, 5, 6, and 7, respectively, in the packet's Requester ID**.
	• 10b. The two msbs of the function number in the Requestor ID are used for Phantom Functions. The device designer may implement functions 0 and 1. When issuing request packets, **Function 0 may also use function numbers 2, 4, and 6 in the packet's Requester ID. Function 1 may also use function numbers 3, 5, and 7 in the packet's Requester ID**. • 11b. All three bits of the function number in the Requestor ID are used for Phantom Functions. The device designer must only implement **Function 0** (and it **may use any function number in the packet's Requester ID**).

5	**Extended Tag Field Supported**. Max supported size of the Tag field when this function acts as a Requester. • 0 = **5-bit Tag field supported** (max of 32 outstanding request per Requester). • 1 = **8-bit Tag field supported** (max of 256 outstanding request per Requester). If 8-bit Tags are supported and will be used, this feature is enabled by setting the Extended Tag Field Enable bit in the Device Control register (see "Device Control Register" on page 905) to one.
8:6	**Endpoint L0s Acceptable Latency**. Acceptable total latency that an Endpoint can withstand due to the transition from the L0s state to the L0 state (see "L0s Exit Latency Update" on page 625). This value is an indirect indication of the amount of the Endpoint's internal buffering. Power management software uses this value to compare against the L0s exit latencies reported by all components in the path between this Endpoint and its parent Root Port to determine whether ASPM L0s entry can be used with no loss of performance. • 000b = Less than 64ns • 001b = 64ns to less than 128ns • 010b = 128ns to less than 256ns • 011b = 256ns to less than 51 ns • 100b = 512ns to less than 1µs • 101b = 1µs to less than 2µs • 110b = 2µs-4µs • 111b = More than 4µs

11:9	**Endpoint L1 Acceptable Latency**. Acceptable latency that an Endpoint can withstand due to the transition from L1 state to the L0 state (see "L1 Exit Latency Update" on page 626). This value is an indirect indication of the amount of the Endpoint's internal buffering. Power management software uses this value to compare against the L1 Exit Latencies reported by all components in the path between this Endpoint and its parent Root Port to determine whether ASPM L1 entry can be used with no loss of performance. • 000b = Less than 1µs • 001b = 1µs to less than 2µs • 010b = 2µs to less than 4µs • 011b = 4µs to less than 8µs • 100b = 8µs to less than 16µs • 101b = 16µs to less than 32µs • 110b = 32µs-64µs • 111b = More than 64µs
12	**Attention Button Present**. When set to one, indicates an Attention Button is implemented on the card or module. Valid for the following PCI Express device Types: • Express Endpoint device • Legacy Express Endpoint device • Switch upstream port • Express-to-PCI/PCI-X bridge
13	**Attention Indicator Present**. When set to one, indicates an Attention Indicator is implemented on the card or module. Valid for the following PCI Express device Types: • Express Endpoint device • Legacy Express Endpoint device • Switch upstream port • Express-to-PCI/PCI-X bridge

14	**Power Indicator Present**. When set to one, indicates a Power Indicator is implemented on the card or module. Valid for the following PCI Express device Types: • Express Endpoint device • Legacy Express Endpoint device • Switch upstream port • Express-to-PCI/PCI-X bridge
25:18	**Captured Slot Power Limit Value** (upstream ports only). In combination with the Slot Power Limit Scale value (see the next row in this table), specifies the upper limit on power supplied by slot: Power limit (in Watts) = Slot Power Limit value x Slot Power Limit Scale value This value is either automatically set by the receipt of a Set Slot Power Limit Message received from the port on the downstream end of the link, or is hardwired to zero. Refer to "Slot Power Limit Control" on page 562 for a detailed description.
27:26	**Captured Slot Power Limit Scale** (upstream ports only). Specifies the scale used for the calculation of the Power Limit (see the previous row in this table): • 00b = 1.0x • 01b = 0.1x • 10b = 0.01x • 11b = 0.001x This value is either automatically set by the receipt of a Set Slot Power Limit Message received from the port on the downstream end of the link, or is hardwired to zero.

Table 21: Device Capabilities Register (RO)

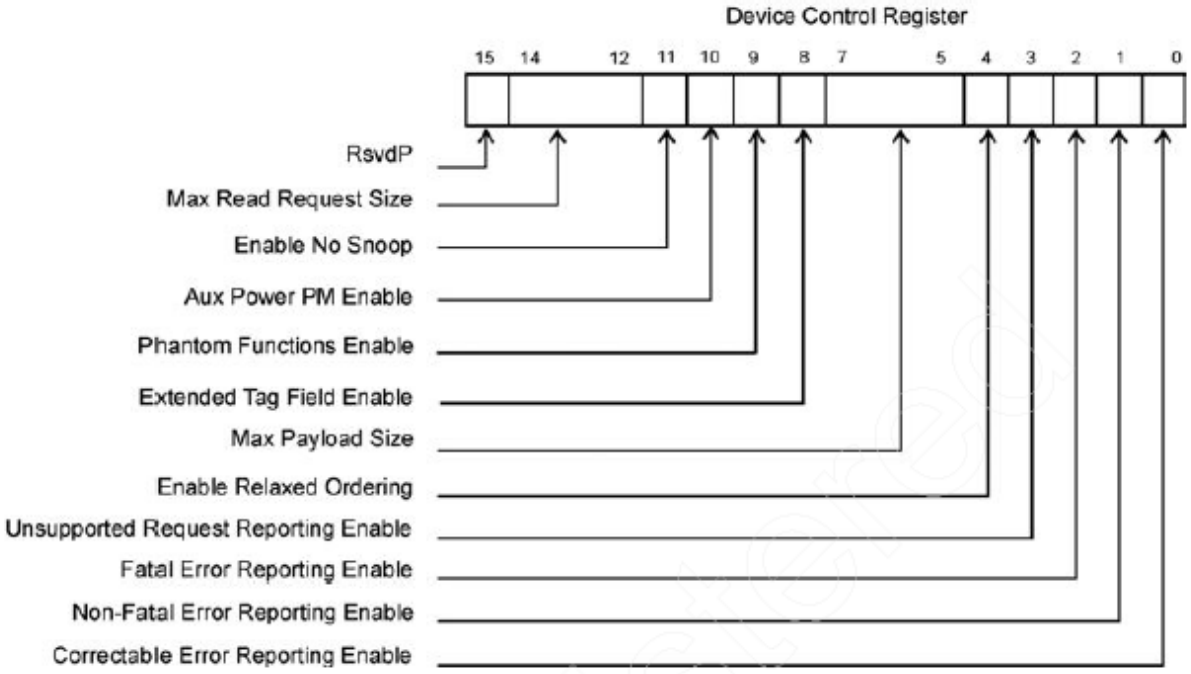


Figure 36: Device Control Register

Bit(s)	Description
0	**Correctable Error Reporting Enable**. For a multifunction device, this bit controls error reporting for all functions. For a Root Port, the reporting of correctable errors occurs internally within the Root Complex. No external ERR_COR Message is generated. Default value of this field is 0.
1	**Non-Fatal Error Reporting Enable.** This bit controls the reporting of non-fatal errors. For a multifunction device, it controls error reporting for all functions. For a Root Port, the reporting of non-fatal errors occurs internally within the Root Complex. No external ERR_NONFATAL Message is generated. Default value of this field is 0.
2	**Fatal Error Reporting Enable.** This bit controls the reporting of fatal errors. For a multifunction device, it controls error reporting for all functions within the device. For a Root Port, the reporting of fatal errors occurs internally within the Root Complex. No external ERR_FATAL Message is generated. Default value of this bit is 0.
3	**Unsupported Request (UR) Reporting Enable.** When set to one, this bit enables the reporting of Unsupported Requests. For a multifunction device, it controls UR reporting for all functions. The reporting of error messages (ERR_COR, ERR_NONFATAL, ERR_FATAL) received by a Root Port is controlled exclusively by the Root Control register (see "Root Control Register" on page 926). Default value of this bit is 0.
4	**Enable Relaxed Ordering.** When set to one, the device is permitted to set the Relaxed Ordering bit (refer to "Relaxed Ordering" on page 319) in the Attributes field of requests it initiates that do not require strong write ordering. Default value of this bit is 1, but it may be hardwired to 0 if a device never sets the Relaxed Ordering attribute in requests it initiates as a Requester.
7:5	**Max Payload Size.** Sets the max TLP data payload size for the device. As a Receiver, the device must handle TLPs as large as the set value; as a Transmitter, the device must not generate TLPs exceeding the set value. Permissible values that can be programmed are indicated by the Max Payload Size Supported in the Device Capabilities register (see "Device Capabilities Register" on page 900).

	• 000b = 128 byte max payload size • 001b = 256 byte max payload size • 010b = 512 byte max payload size • 011b = 1024 byte max payload size • 100b = 2048 byte max payload size • 101b = 4096 byte max payload size • 110b = Reserved • 111b = Reserved Default value of this field is 000b.
8	**Extended Tag Field Enable.** When set to one, enables a device to use an 8-bit Tag field as a requester. If cleared to zero, the device is restricted to a 5-bit Tag field. Also refer to the description of the Phantom Functions Supported field in Table 24 - 2 on page 901. The default value of this bit is 0. Devices that do not implement this capability hardwire this bit to 0.
9	**Phantom Functions Enable**. See the description of the Phantom Functions Supported field in Table 24 - 2 on page 901. Default value of this bit is 0. Devices that do not implement this capability hardwire this bit to 0.
10	**Auxiliary (AUX) Power PM Enable.** When set to one, this bit enables a device to draw Aux power independent of PME Aux power. In a legacy OS environment, devices that require Aux power should continue to indicate PME Aux power requirements. Aux power is allocated as requested in the Aux Current field of the Power Management Capabilities register (PMC; see "Auxiliary Power" on page 645), independent of the PME Enable bit in the Power Management Control/Status register (PMCSR; see "PM Control/Status (PMCSR) Register" on page 599). For multifunction devices, a component is allowed to draw Aux power if at least one of the functions has this bit set. • *Note*: Devices that consume Aux power must preserve the value in this field when Aux power is available. In such devices, this register value is not modified by hot, warm, or cold reset. • Devices that do not implement this capability hardwire this bit to 0.

11	**Enable No Snoop.** Software sets this bit to one if the area of memory this Requester will access is not cached by the processor(s). When a request packet that targets system memory is received by the Root Complex (i.e., the memory that the processors cache from), the Root Complex does not have to delay the access to memory to perform a snoop transaction on the processor bus if the No Snoop attribute bit is set. This speeds up the memory access. • Note that setting this bit to one should not cause a function to unequivocally set the No Snoop attribute on every memory requests that it initiates. The function may only set the bit when it knows that the processor(s) are not caching from the area of memory being accessed. • Default value of this bit is 1 and it may be hardwired to 0 if a device never sets the No Snoop attribute in Request transactions that it initiates.
14:12	**Max_Read_Request_Size.** Max read request size for the device when acting as the Requester. The device must not generate read requests with a size > this value. • 000b = 128 byte max read request size • 001b = 256 byte max read request size • 010b = 512 byte max read request size • 011b = 1KB max read request size • 100b = 2KB max read request size • 101b = 4KB max read request size • 110b = Reserved • 111b = Reserved Devices that do not generate read requests larger than 128 bytes are permitted to implement this field as Read Only (RO) with a value of 000b. Default value of this field is 010b.

Table 22: Device Control Register (R/W)

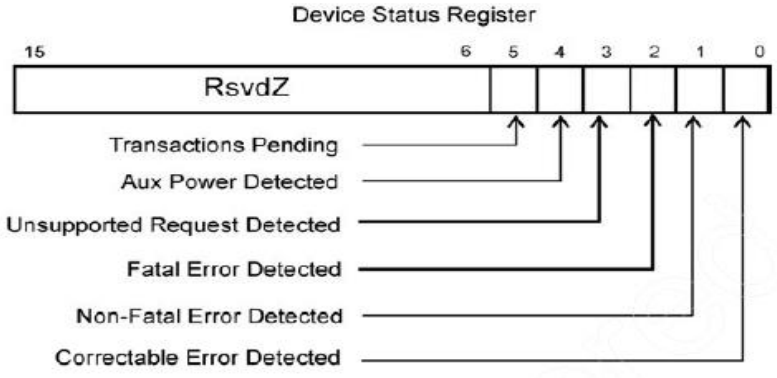


Figure 37: Device Status Register

Bit(s)	Type	Description
0	RW1C	**Correctable Error Detected**. A one indicates that one or more correctable errors were detected since the last time this bit was cleared by software. Correctable errors are reflected by this bit regardless of whether error reporting is enabled or not in the Device Control register (see "Device Control Register" on page 905). In a multifunction device, each function indicates whether or not that function has detected any correctable errors using this bit. For devices supporting Advanced Error Handling (see "Advanced Error Reporting Mechanisms" on page 382), errors are logged in this register regardless of the settings of the Correctable Error Mask register. Default value of this bit is 0.
1	RW1C	**Non-Fatal Error Detected**. A one indicates that one or more non-fatal errors were detected since the last time this bit was cleared by software. Non-fatal errors are reflected in this bit regardless of whether error reporting is enabled or not in the Device Control register (see "Device Control Register" on page 905). In a multifunction device, each function indicates whether or not that function has detected any non-fatal errors using this bit. For devices supporting Advanced Error Handling, errors are logged in this register regardless of the settings of the Uncorrectable Error Mask register (*note that the 1.0a spec says "Correctable Error Mask register", but the authors think this is incorrect*). Default value of this bit is 0.
2	RW1C	**Fatal Error Detected**. A one indicates that one or more fatal errors were detected since the last time this bit was cleared by software. Fatal errors are reflected in this bit regardless of whether error reporting is enabled or not in the Device Control register (see "Device Control Register" on page 905). In a multifunction device, each function indicates whether or not that function has detected any fatal errors using this bit. For devices supporting Advanced Error Handling (see "Advanced Error Reporting Capability" on page 930), errors are logged in this register regardless of the settings of the Uncorrectable Error Mask register (*note that the 1.0a spec erroneously says "Correctable Error Mask register."*) Default value of this bit is 0.

3	RW1C	**Unsupported Request (UR) Detected**. When set to one, indicates that the function received an Unsupported Request. Errors are reflected in this bit regardless of whether error reporting is enabled or not in the Device Control register (see "Device Control Register" on page 905). In a multifunction device, each function indicates whether or not that function has detected any UR errors using this bit. Default value of this field is 0.
4	RO	**Aux Power Detected**. Devices that require Aux power set this bit to one if Aux power is detected by the device.
5	RO	**Transactions Pending**. When set to one, indicates that this function has issued non-posted request packets which have not yet been completed (either by the receipt of a corresponding Completion, or by the Completion Timeout mechanism). A function reports this bit cleared only when all outstanding non-posted requests have completed or have been terminated by the Completion Timeout mechanism. • **Root and Switch Ports**: Root and Switch Ports adhering solely to the 1.0a Express spec never issue non-posted requests on their own behalf. Such Root and Switch Ports hardwire this bit to 0b.

Table 23: Device Status Register

Link Registers gives more Capabilities and control and status.

MSI Capability

A PCI Express function indicates its support for MSI via the MSI Capability registers. Each native PCI Express function must implement a single MSI register set within its own configuration space. Note that the PCI Express specification defines two register formats:

- 64-bit memory addressing format required by all native PCI Express devices and optionally implemented by Legacy end-points.
- 32-bit memory addressing format optionally supported by Legacy endpoints.

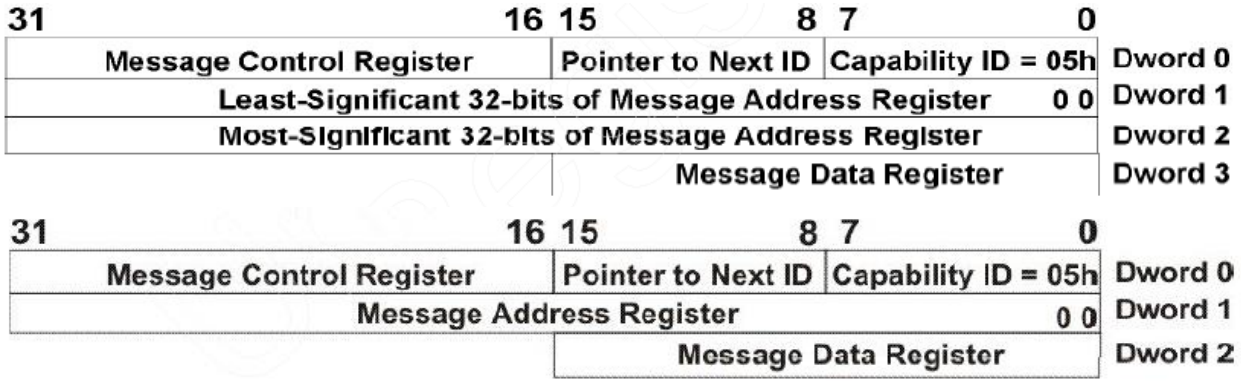


Figure 38: MSI Capability Register Set 32&64-bit

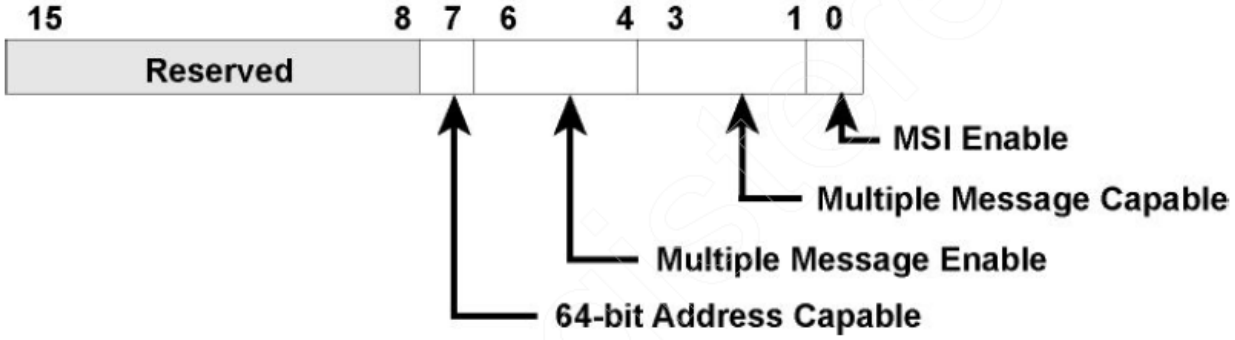


Figure 39: Message Control Register

Bit(s)	Field Name	Description
15:8	Reserved	Read-Only. Always zero.
7	64-bit Address Capable	Read-Only. • 0 = Function does not implement the upper 32-bits of the Message Address register and is incapable of generating a 64-bit memory address. • 1 = Function implements the upper 32-bits of the Message Address register and is capable of generating a 64-bit memory address.
6:4	Multiple Message Enable	Read/Write. After system software reads the Multiple Message Capable field (see next row in this table) to determine how many messages are requested by the device, it programs a 3-bit value into this field indicating the actual number of messages allocated to the device. The number allocated can be equal to or less than the number actually requested. The state of this field after reset is 000b. The field is encoded as follows: **Value** — **Number of Messages Requested** 000b — 1 001b — 2 010b — 4 011b — 8 100b — 16 101b — 32 110b — Reserved 111b — Reserved
3:1	Multiple Message Capable	Read-Only. System software reads this field to determine how many messages the device would like allocated to it. The requested number of messages is a power of two, therefore a device that would like three messages must request that four messages be allocated to it. The field is encoded as follows:

		Value / **Number of Messages Requested** 000b — 1 001b — 2 010b — 4 011b — 8 100b — 16 101b — 32 110b — Reserved 111b — Reserved
0	MSI Enable	Read/Write. State after reset is 0, indicating that the device's MSI capability is disabled. • **0** = Function is **disabled** from using **MSI**. It must use INTX Messages to deliver interrupts (legacy endpoint or bridge). • **1** = Function is **enabled** to use **MSI** to request service and is forbidden to use its interrupt pin.

Table 24: Message Control Register

Message Address Register → The lower two bits of the 32-bit Message Address register are hardwired to zero and cannot be changed. In other words, the address assigned by system software is always aligned on a DW address boundary.

The upper 32-bits of the Message Address register are required for native PCI Express devices and optional for legacy endpoints. This register is present if Bit 7 of the Message Control register is set. If present, it is a read/write register and it is used in conjunction with the Message Address register to assign a 32-bit or a 64-bit memory address to the device:

- If the upper 32-bits of the Message Address register are set to a non-zero value by the system software, then a 64-bit message address has been assigned to the device using both the upper and lower halves of the register.
- If the upper 32-bits of the Message Address register are set to zero by the system software, then a 32-bit message address has been assigned to the device using both the upper and lower halves of the register.

Message Data Register → The system software assigns the device a base message data pattern by writing it into this 16-bit, read/write register. When the device must generate an interrupt request, it writes a 32-bit value to the memory address specified in the Message Address register. The data written has the following format:

- The upper 16 bits are always set to zero.
- The lower 16 bits are supplied from the Message Data register. If more than one message has been assigned to the device, the device modifies the lower bits (the number of modifiable bits depends on how many messages have been assigned to the device by

the configuration software) of the data from the Message Data register to form the appropriate message for the event it wishes to report to its driver.

3-5-5 Extended Capabilities

Capabilities extends the configuration space to 64 DW but it wasn't enough so it was extended to 1024 DW.

Advanced Error Reporting Extended Capability (AER)

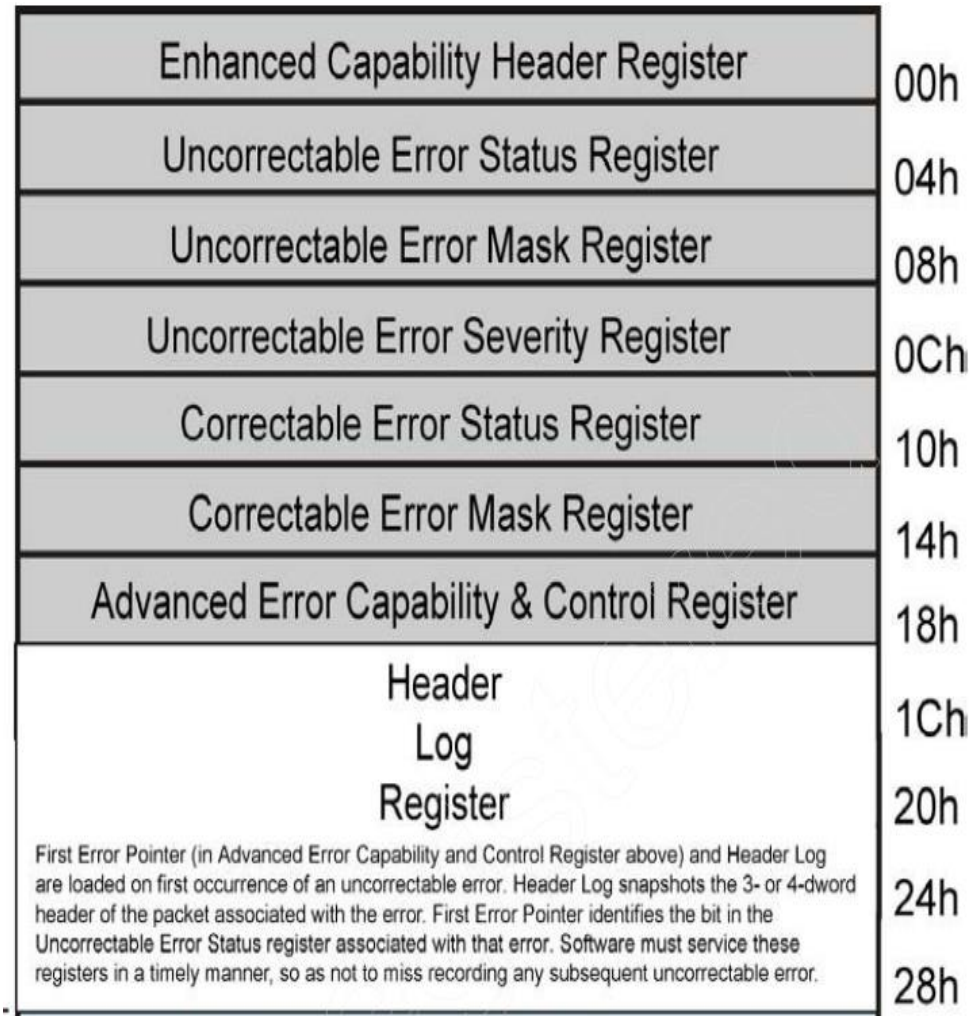


Figure 40: AER Extended Capability Register Set

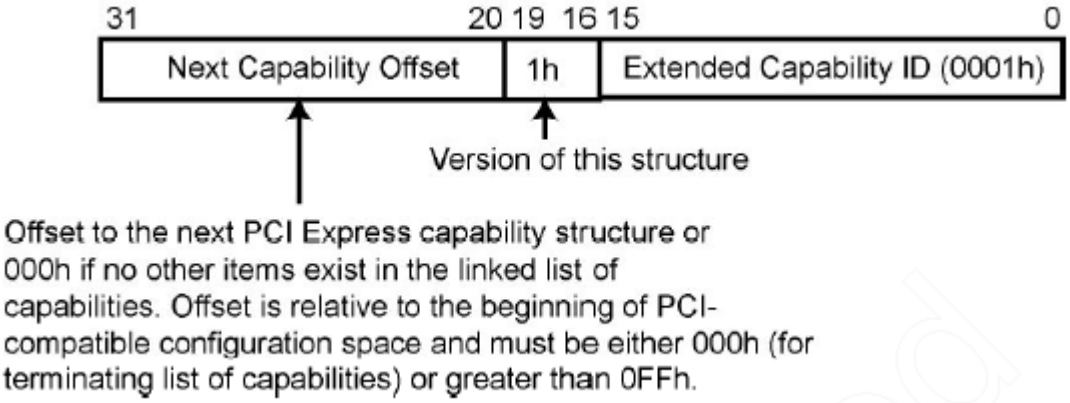


Figure 41: AER Enhanced Capability Register

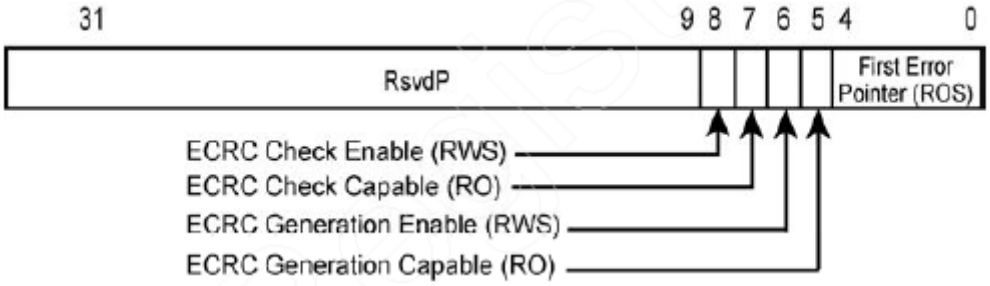


Figure 42: Advanced Error Capabilities and Control Register

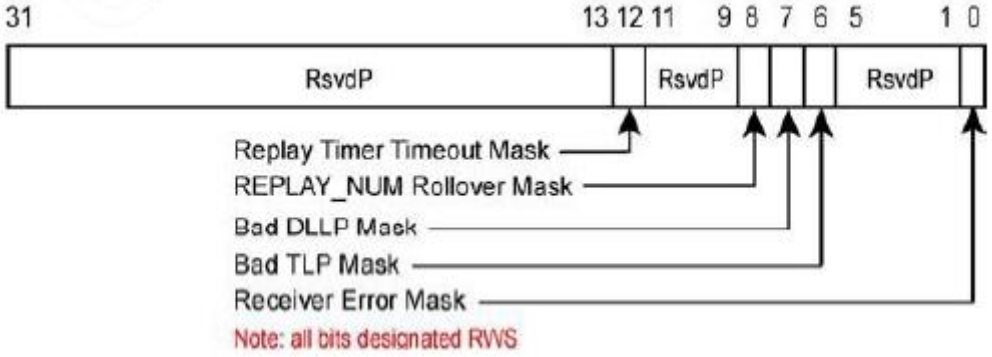


Figure 43: Advanced Error Correctable Error Mask Register

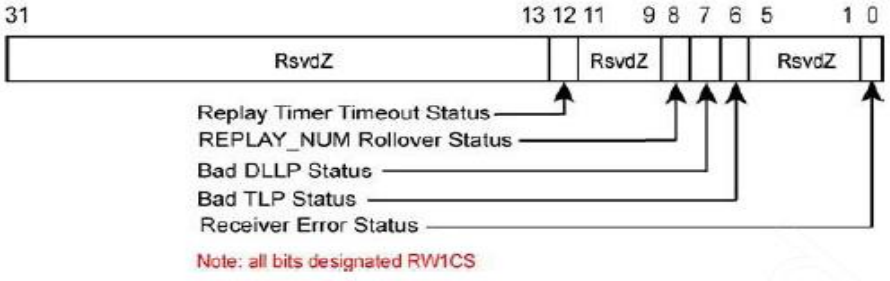


Figure 44: Advanced Error Correctable Error Status Register

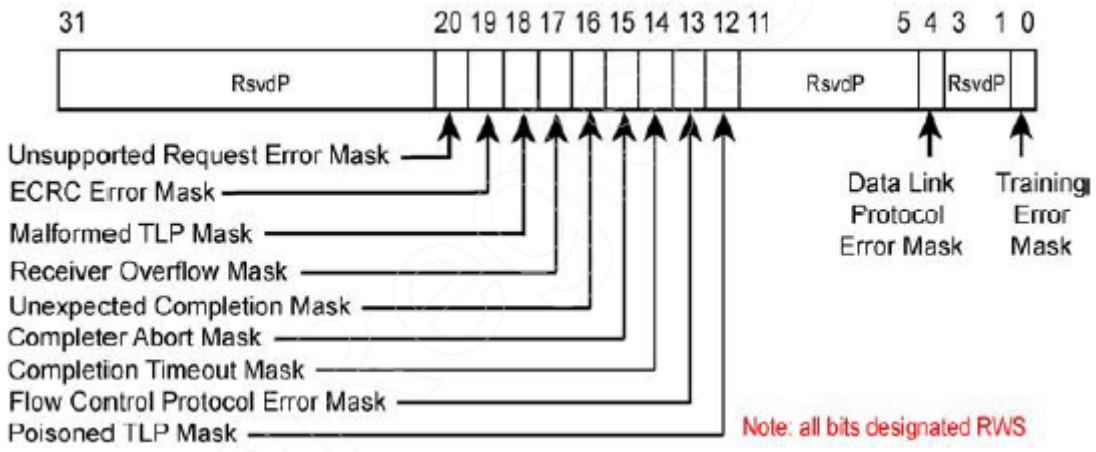


Figure 45: Advanced Error Uncorrectable Error Mask Register

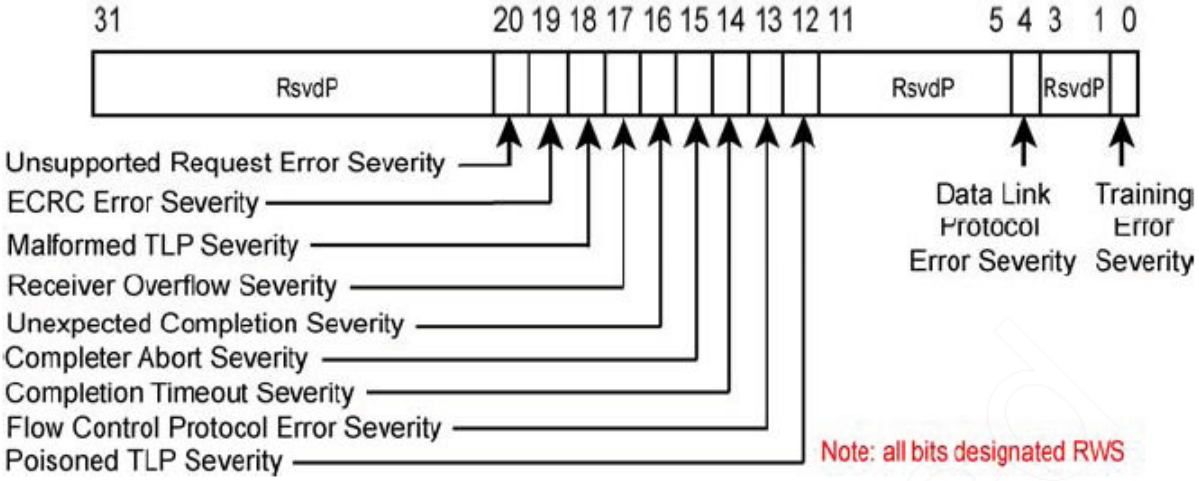


Figure 46: Advanced Error Uncorrectable Error Severity Register

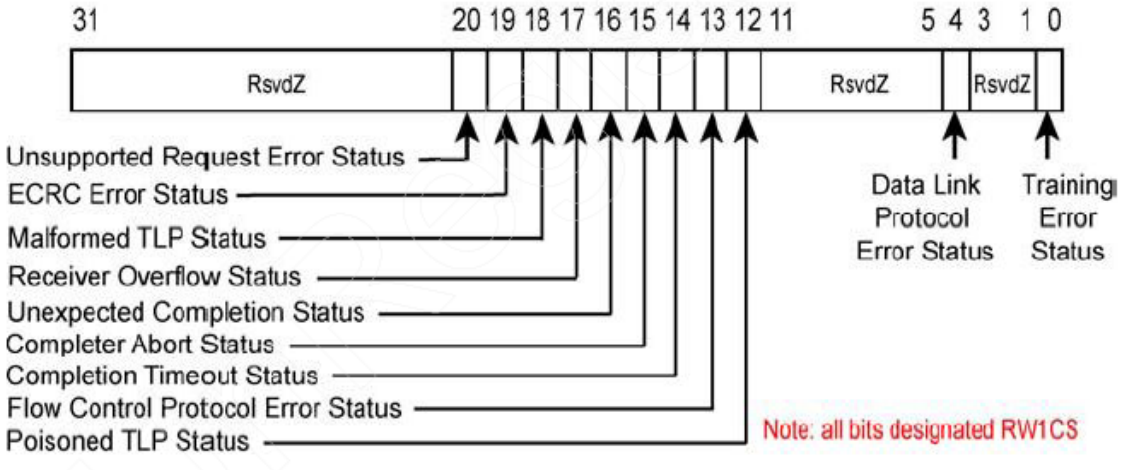


Figure 47: Advanced Error Uncorrectable Error Status Register

Register Group	Register	Description
NA	Enhanced Capability Header	Capability ID = 0001h. The Capability Version field in this register is assigned by the SIG and defines the layout of the register set. It must be 1h for all of the extended capabilities currently defined. See Figure 24-17 on page 935.
NA	Capabilities and Control Register	Contains the following bits fields: • **First Error Pointer**. Read-only. Identifies the bit position of the first error reported in the Uncorrectable Error Status register (see Figure 24-23 on page 937). • **ECRC Generation Capable**. Read-only. 1 indicates that the function is capable of generating ECRC (End-to-End CRC; refer to "ECRC Generation and Checking" on page 361). • **ECRC Generation Enable**. Read/write sticky bit. When set to one, enables ECRC generation. Default = 0. • **ECRC Check Capable**. Read-only. 1 indicates that the function is capable of checking ECRC. • **ECRC Check Enable**. Read/write sticky bit. When set to one, enables ECRC checking. Default = 0.
Correctable Error Registers	Correctable Error Mask Register	Controls the reporting of individual correctable errors by the function to the Root Complex via a PCI Express error message. A masked error (respective bit set to one) is not reported to the Root Complex by the function. This register contains a mask bit for each corresponding error bit in the Correctable Error Status register (see the next row in this table and Figure 24-19 on page 935).
	Correctable Error Status Register	Reports the error status of the function's correctable error sources. Software clears a set bit by writing a 1 to the respective bit. See Figure 24-20 on page 936.

Uncorrectable Error Registers	Uncorrectable Error Mask Register	Controls the function's reporting of errors to the Root Complex via a PCI Express error message. A masked error (respective bit set to 1b): • is not logged in the Header Log register (see Figure 24-16 on page 931), • does not update the First Error Pointer (see the description of the Capabilities and Control Register in this table), and • is not reported to the Root Complex. This register (see Figure 24-21 on page 936)contains a mask bit for each corresponding error bit in the Uncorrectable Error Status register.
	Uncorrectable Error Severity Register	Each respective bit controls whether an error is reported to the Root Complex via a non-fatal or fatal error message. An error is reported as fatal if the corresponding bit is set to one. See Figure 24-22 on page 937.
	Uncorrectable Error Status Register	Reports the error status of the function's uncorrectable error sources. See Figure 24-23 on page 937.

Table 25: AER Extended Capabilities Register Set

3-5-6 Our Design

Configuration Space is implemented as a class inside transaction layer with interfacing read and write APIs with taking in consideration inside the class the read only and read write bits and capabilities. Inside the class it's an array with all default values at constructor then its configured by software during device driver phase with series of configuration read and write packets then after device driver the thread connects it to other blocks which needs configuration and writes to it and also respond to normal configuration reads and writes.

3-6 TLP Decomposition

3-6-1 Block Interfacing

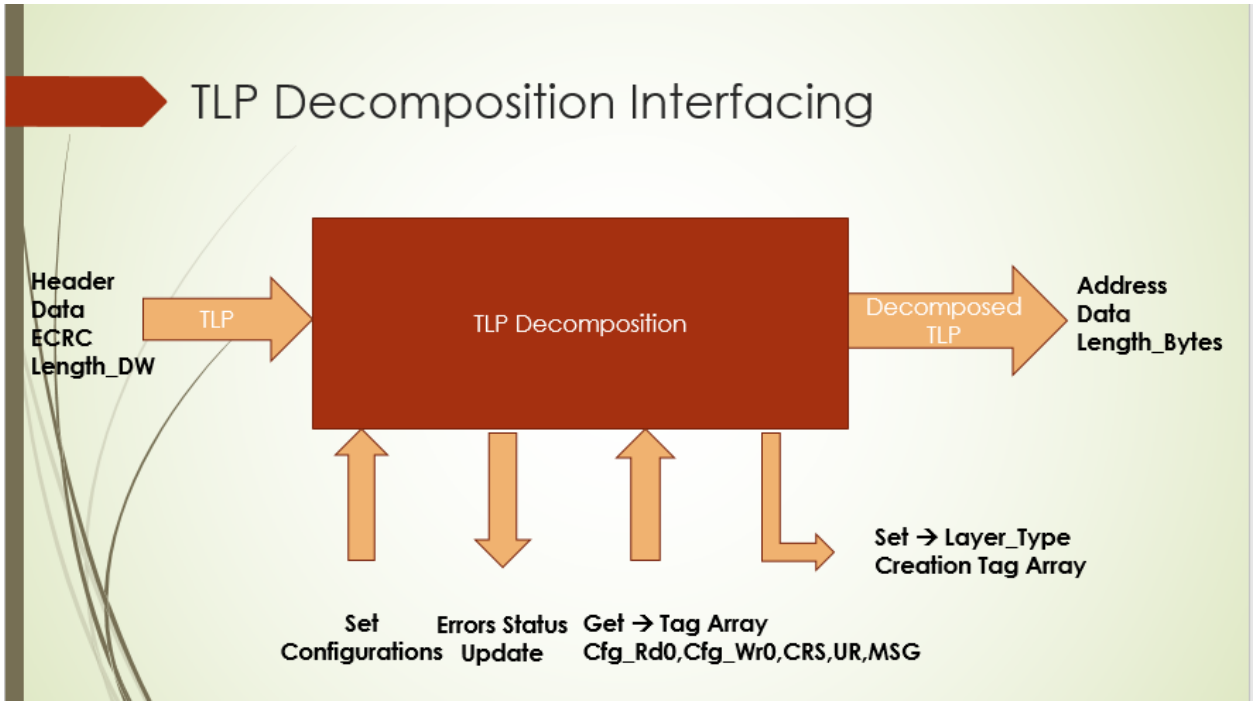


Figure 48: TLP Decomposition Interfacing

As an input to decomposition block (TLP):

- Header
- Data in DWs
- ECRC if exist
- Length of Data in DWs

Output is decomposed TLP to the Application layer:

- Address
- Data
- Length of Data in Bytes

As interfacing with thread:

- Set needed Configurations → Enables for errors detections and handling and enables for decoding and commands like memory space decode enable which enables the layer to decode memory packets
- Get Error update status → after detection and handling the error sends the bits to be updated in configuration space to thread and thread to configuration space
- Get Tag Array and Setting the other one from the creation → Tag array used between creation and decomposition to save headers awaiting completions to be able to decode completions in decomposition block and create completions in creation block
- Setting Layer type → Endpoint or Bridge to decode correctly

- Telling the thread to create some packets as completions for CfgRd & CfgWr and UR and CRS and MSG

3-6-2 Block Flow

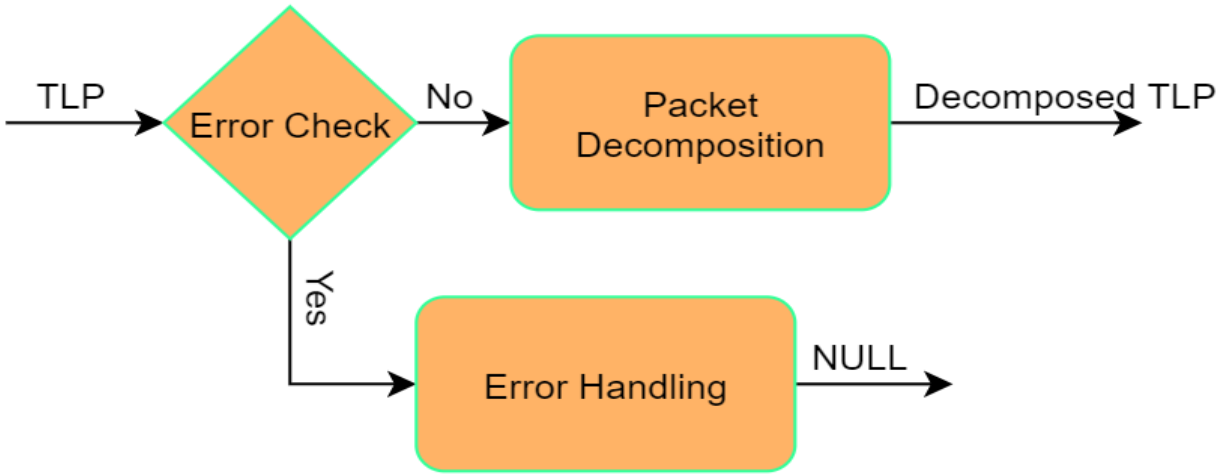


Figure 49: TLP Decomposition Block Flow

First after receiving TLP check for errors if any detected then go for error handling block then updating thread but if no errors detected it go for packet decomposition block to decompose the packet and return it application layer.

3-6-3 Error Checking

Unsupported Request UR

- Max Payload → Receiving packet exceeds device max payload supported and enabled
- Wrong Format
- Unsupported Type
- BAR → wrong address
- Memory Address Decoder → Receiving memory packet while memory address decoding enable is disabled
- Wrong Completer ID

Poisoned TLP

- EP Check → poisoned data bit is set ECRC error
- ECRC Check

Malformed TLP

- Length Check → length inside the packet doesn't match with length which si input to decomposition block
- Digest bit → if set and no ECRC and vice versa
- Byte Enable Violations
- Unknown Type

Completion Abort CA

- Received Completion with Completion Status CA

Unexpected Completion

- Tag Mismatch

3-6-4 Error handling Block

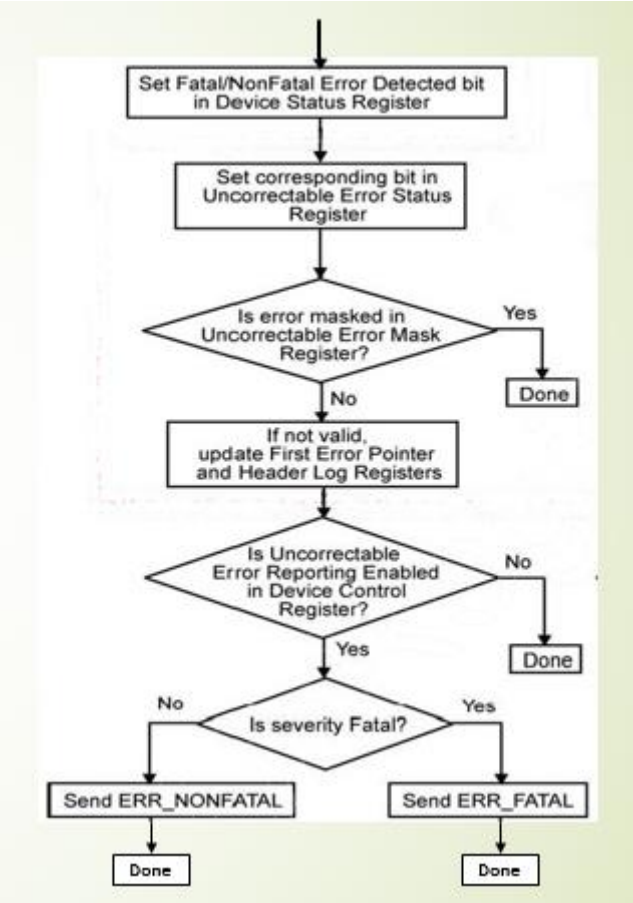


Figure 50: Error Handling Flowchart

Error handling block executes the flow chart above by setting some bits in configuration space and reading some bits by thread to handle the error correctly according to specification.

3-6-5 Packet Decomposition

Memory Packets

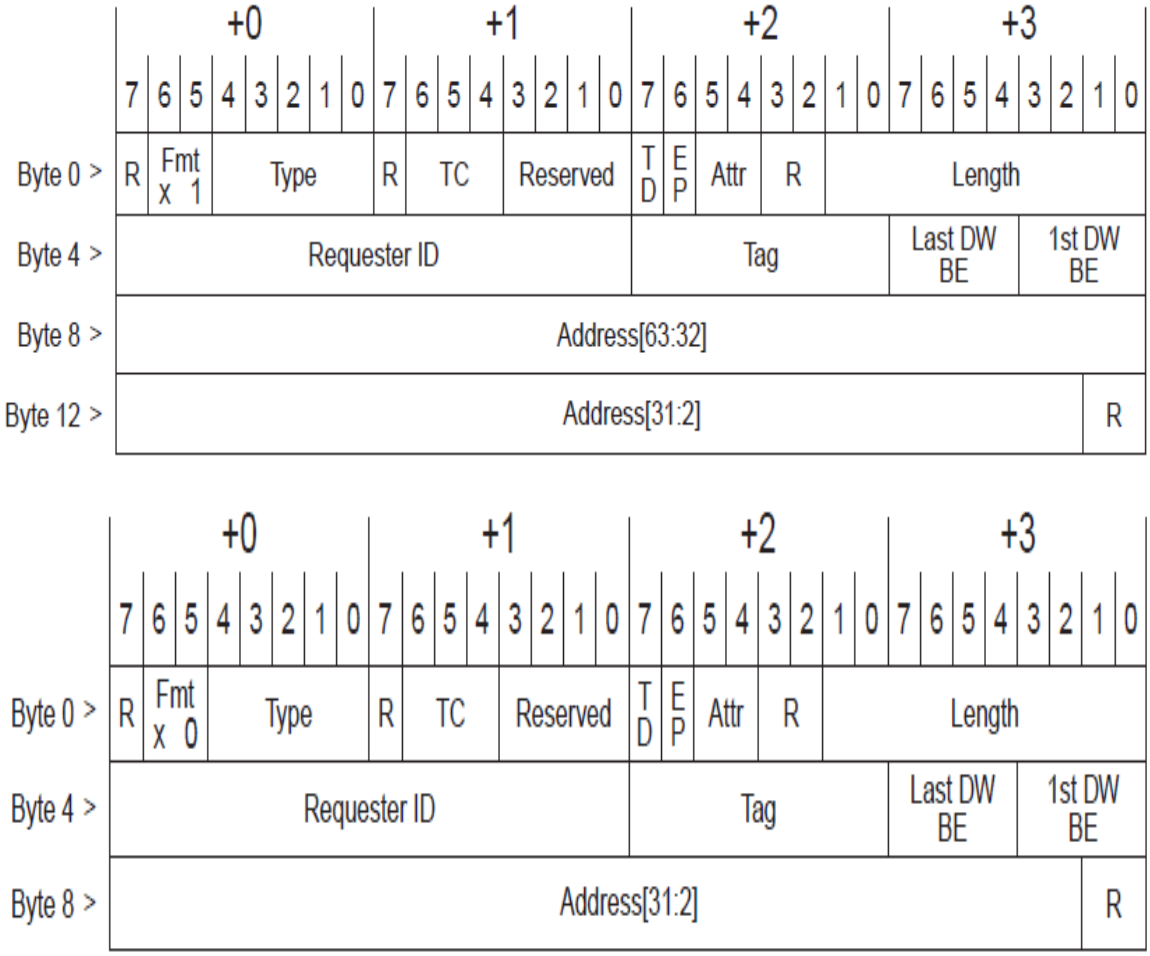


Figure 51: Memory TLPs

Memory packets (MRd32, MRd64, MWr32, MWr64) decomposed as follows:

- Address Calculation by byte enable and DW aligned address
- Data Length Bytes Calculation by byte enable and Length DW field
- Data Calculation by byte enable (in case of MWr)
- Header is Saved in Tag Array in case of Memory Read for Completion (Tag & ID)

Configuration Packets

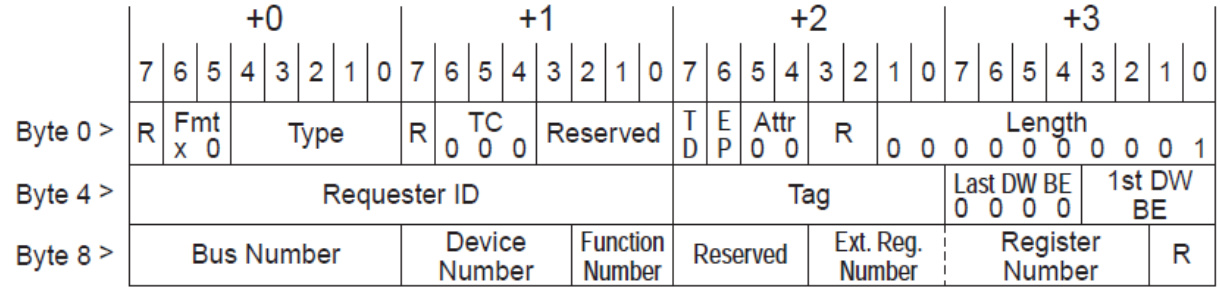


Figure 52: Configuration TLPs

Configuration Packets (CfgRd, CfgWr) decomposed as follows:

- Register Number (Address) Calculation by byte enable and DW aligned Register Number
- Data Length Bytes Calculation by byte enable
- Data Calculation by byte enable (in case of CfgWr)
- Header is Saved in Tag Array for Completion (Tag & ID)

Completion Packets

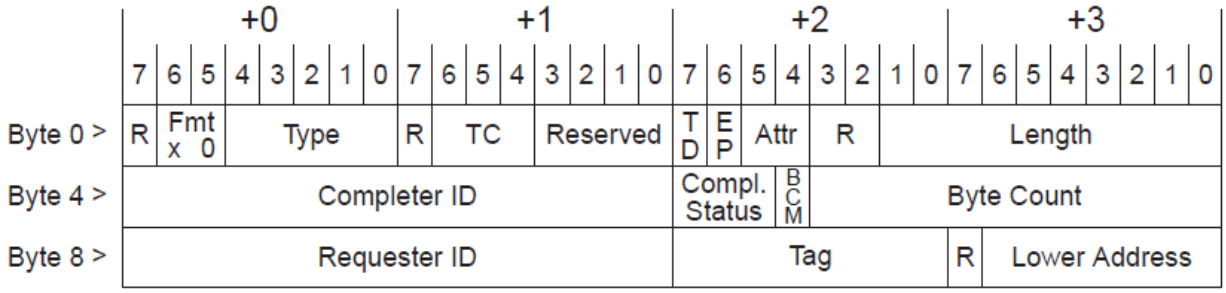


Figure 53: Completions TLPs

Completion Packets (Cpl, CplD) decomposed as follows:

- Completion Status → UR, CA, CRS, SC
- UR, CA → Generate Error & Clear Header from Tag Array & Update Application Layer (Device Core)
- CRS → Send to thread to retry the Request
- SC
    - Cpl – CfgWr
        - Clear Tag Array
    - CplD – CfgRd
        - Register Number (Address) Calculation by byte enable and DW aligned Register Number from the Tag Array
        - Data Length Bytes Calculation by byte enable
        - Data Calculation by byte enable
        - Clear Header from Tag Array

- CplD – MRd
  - Address Calculation by byte enable and DW aligned Address from the Tag Array
  - Data Length Bytes Calculation by byte enable and DW aligned address
  - Data Calculation by byte enable and DW aligned address
  - Clear Header from Tag Array
  - In Case of Split Transaction (Multiple Completions) Data is saved in Decomposition till completed before doing above steps

3-6-6 Implementation

TLP Decomposition was implemented as a class with APIs for interfacing with it to decompose the packet and connect with thread to get data from it and to tell the thread to do something like sending packets and update errors.

3-7 Thread

3-7-1 Operations

Thread mainly used as a controller to connect between blocks by polling and Controlling the flow in the transaction layer and servicing blocks

- Block connections
  - Taking packets from Rx Buffer to TLP Decomposition then to Transaction to Application Queue
  - Taking packets from Creation to thread Queue & from Thread Queue to Tx Buffer by Round Robin
  - Taking packets from Tx Buffer to DataLink Layer if Flow Control Logic is true
  - Taking Decomposed packets from Transaction to Application Queue by Set Callback
  - Get Flow Control Variables from DataLink

- Servicing Blocks
  - Taking Tag Array from TLP decomposition to TLP Creation and Vice Versa
  - Set Needed Configurations to Configuration space to TLP Creation and Decomposition
  - Update Configurations from TLP Creation and Decomposition to Configuration space
  - Creating and Sending packets required by TLP Decomposition (CfgRd, CfgWr, CRS, UR, MSG)

- Control
  - Checking Completion Timeout and executing Error handling flow chart in case of error
  - Calling DataLink Update (DataLink Controller)

3-7-2 Implementation

The thread was implemented as a private function inside transaction layer and the thread is created in the transaction layer block constructor and terminated in destructor. It's an infinite loop which use polling by checking all blocks APIs to interface with them and give them what they need and take from them what the thread needs and connecting between them by taking data from one block to another. And calling Datalink update (thread) and datalink flow control packets.

3-8 Flow Control

3-8-1 Concept

The ports at each end of every PCI Express link must implement Flow Control. Before a transaction packet can be sent across a link to the receiving port, the transmitting port must verify that the receiving port has sufficient buffer space to accept the transaction to be sent. In many other architectures including PCI and PCI-X, transactions are delivered to a target device without knowing if it can accept the transaction. If the transaction is rejected due to insufficient buffer space, the transaction is resent (retried) until the transaction completes. This procedure can severely reduce the efficiency of a bus, by wasting bus bandwidth when other transactions are ready to be sent.

Because PCI Express is a point-to-point implementation, the Flow Control mechanism would be ineffective, if only one transaction stream was pending transmission across a link. That is, if the receive buffer was temporarily full, the transmitter would be prevented from sending a subsequent transaction due to transaction ordering requirements, thereby blocking any further transfers. PCI Express improves link efficiency by implementing multiple flow control buffers for separate transaction streams (virtual channels). Because Flow Control is managed separately for each virtual channel implemented for a given link, if the Flow Control buffer for one VC is full, the transmitter can advance to another VC buffer and send transactions associated with it.

The link Row Control mechanism uses a credit-based mechanism that allows the transmitting port to check buffer space availability at the receiving port. During initialization each receiver reports the size of its receive buffers (in Flow Control credits) to the port at the opposite end of the link. The receiving port continues to update the transmitting port regularly by transmitting the number of credits that have been freed up. This is accomplished via Flow Control DLLPs.

Flow control logic is located in the transaction layer of the transmitting and receiving devices. Both transmitter and receiver sides of each device are involved in flow control.

- Devices Report Buffer Space Available: The receiver of each node contains the Flow Control buffers. Each device must report the amount of flow control buffer space they have available to the device on the opposite end of the link. Buffer space is reported in units called Flow Control Credits (FCCs). The number of Flow Control Credits within each buffer is for-warded from the transaction layer to the transmit side of the link layer. The link creates a Flow Control DLLP that carries this credit information to the receiver at the opposite end of the link. This is done for each Flow Control Buffer.
- Receiving Credits: The receiver also receives Flow Control DLLPs from the device at the opposite end of the link. This information is transferred to the transaction layer to update the Flow Control Counters that track the amount of Row Control Buffer space in the other device.
- Credit Checks Made: Each transmitter checks consults the Flow Control Counters to check available credits. If sufficient credits are available to receive the transaction pending delivery, then the transaction is forwarded to the link layer and is ultimately sent to the opposite device. If enough credits are not available, the transaction is temporarily blocked until additional Flow Control credits are reported by the receiving device.

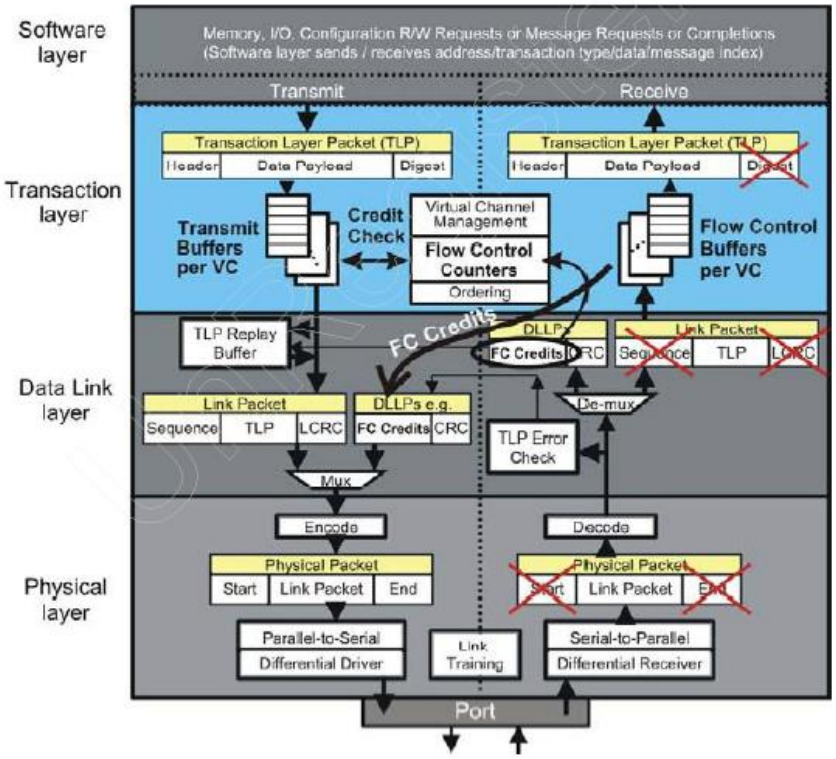


Figure 54: Flow Control Logic Location

3-8-2 Flow Control Buffers

Tx & Rx Buffers are divided to six Buffers as shown in the figure to easy calculate credits for counters and to separate headers from data as there is packets without data so check for buffer of data only.

- Posted Headers PH → MWr , MSG
- Posted Data PD → MWr
- Non-Posted Headers NPH → CfgRd , CfgWr , MRd
- Non-Posted Data NPD → CfgWr
- Completion Headers CPLH → Cpl , CplD
- Completion Data CPLD → CplD

Header Credit for PH & NPH is 5 DW

Header Credit for CPLH is 4 DW

Data Credit for PD & NPD & CPLD is 4 DW

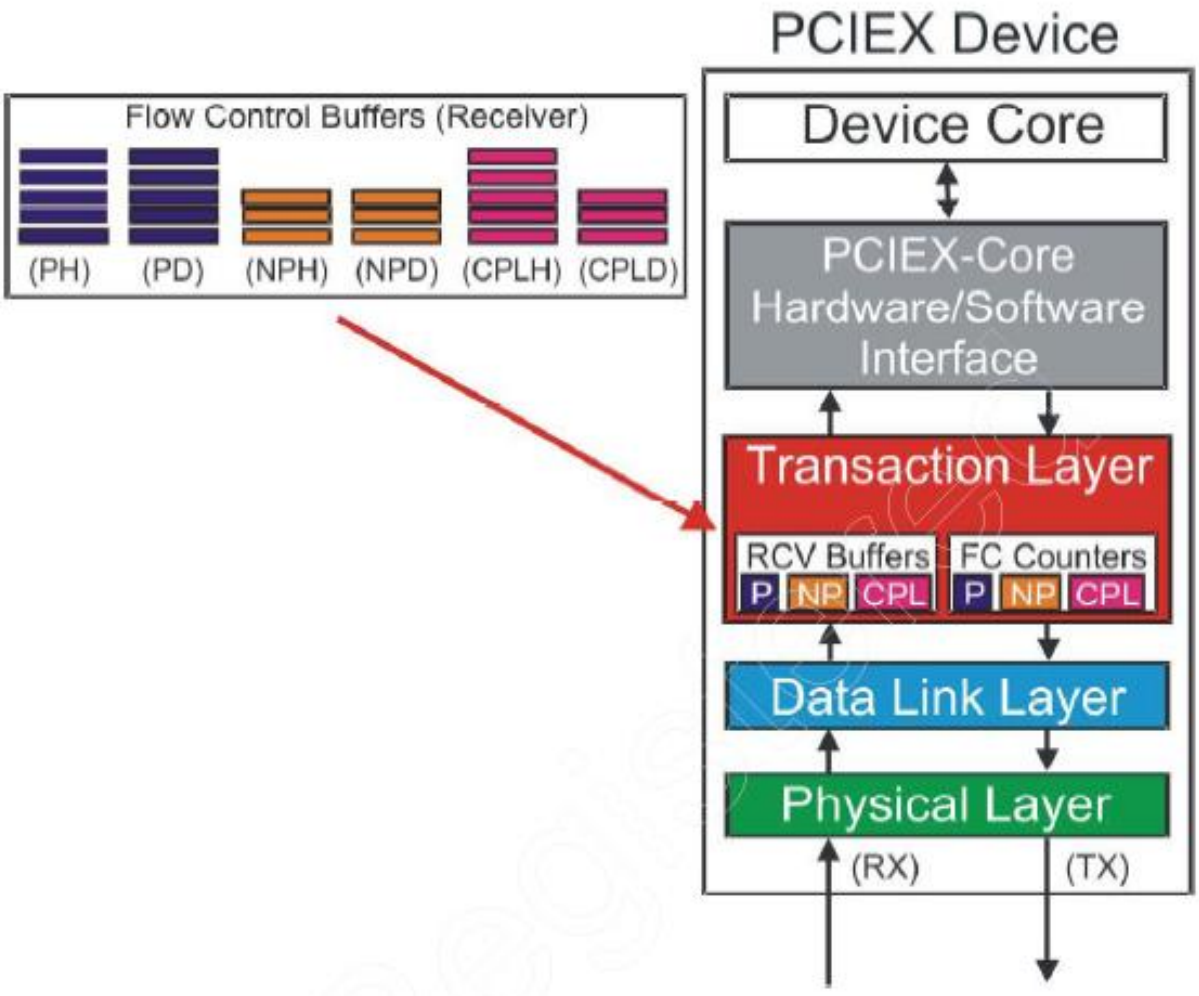


Figure 55: Flow Control Buffers

3-8-3 Flow Control Logic

Transmitter Elements

- Pending Transaction Buffer → holds transactions that are pending transfer within the same virtual channel.
- Credit Consumed Counter → tracks the size of all transactions sent from the VC buffer (of the specified type, e.g., non-posted headers) in Flow Control credits. This count is abbreviated "CC."
- Credit limit Register → this register is initialized by the receiving device when it sends Flow Control initialization packets to report the size of the corresponding Flow Control receive buffer. Following initialization, Flow Control update packets are sent periodically to add more Flow Control credits as they become available at the receiver. This value is abbreviated
- Flow Control Gating Logic → performs the calculations to determine if the receiver has sufficient Row Control credits to receive the pending TLP (PTLP). In essence, this check ensures that the total CREDITS CONSUMED (CC) plus the credit required for the next packet pending transmission (PTLP) does not exceed the CREDIT LIMIT (CL). This specification defines the following equation for performing the check, with all values represented in credits:

$$CL - (CC + PTLP) mod 2^{[FieldSize]} \leq 2^{[FieldSize]}/2$$

Receiver Elements

- Flow Control (Receive) Buffer → stores incoming header or data information.
- Credit Allocated → This counter tracks the total Flow Control credits that have been allocated (made available) since initialization. It is initialized by hardware to reflect the size of the associated Flow Control buffer. As the buffer fills the amount of available buffer space decreases until transactions are removed from the buffer. The number of Row Control credits associated with each transaction removed from the buffer is added to the CREDIT_ALLOCATED counter; thereby keeping a running count of new credits made available.
- Credits Received Counter (optional) — this counter keeps track of the total size of all data received from the transmitting device and placed into the Flow Control buffer (in Flow Control credits). When flow control is functioning properly, the CREDITS_RECEIVED count should be the same as CREDITS_CONSUMED count at the transmitter and be equal to or less than the CREDIT_ALLOCATED count. If this is not true, then a flow control buffer overflow has occurred and error is detected. Although optional the specification recommends its use. The error checking equation is as shown.

$$(CA - CR) mod 2^{[FieldSize]} > 2^{[FieldSize]}/2$$

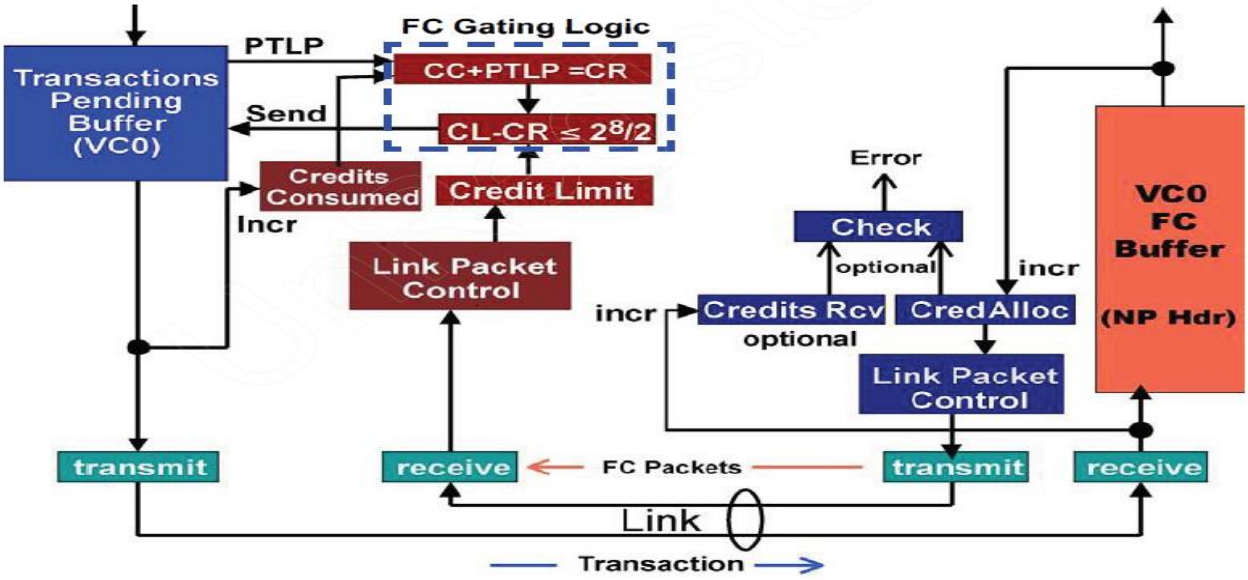


Figure 56: Flow Control Logic Elements

3-8-4 Implementation

Flow control logic is implemented in transaction and counters are updated continuously by datalink layer through APIs called by thread. The buffers is made as a queue of credits 6 at transmitter and 6 at receiver. The counters are private variables in the transaction layer. The flow control uses these counters when the transaction has packets ready to be transferred to datalink layer, then it applies the flow control gating equation shown above in the Flow Control Elements to make sure that the receiver has enough space to receive the packets.

Chapter 4

Datalink Layer

4-1 Data Link Layer Overview

Data Link Layer is the middle PCI-Express architectural layer and interacts with both the Physical Layer and the Transaction Layer. Its primary responsibility is data integrity, including error detection and error correction and to provide a reliable mechanism for exchanging Transaction Layer Packets (TLPs) between the two components on a Link
As well as link management support

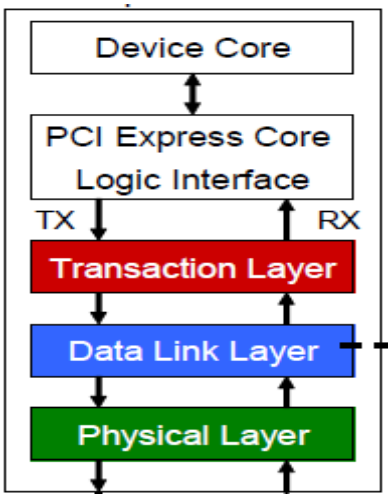


Figure 57: Data link layer overview

4-2 Data Integrity in Data Link Layer

The Data Link Layer takes the TLPs from the transmit side of the Transaction Layer and it adds a sequence number to the front of the packet and a LCRC error checker to the tail The Receive Data Link Layer validates received TLPs by checking the Sequence Number, LCRC code and any error indications from the Receive Physical Layer. this ensures that the receiver transaction layer should receive TLP in the same order that the transmitter send them .so this thing enhance QOS. In case of error in a TLP, Data Link Layer Retry is used for recovery

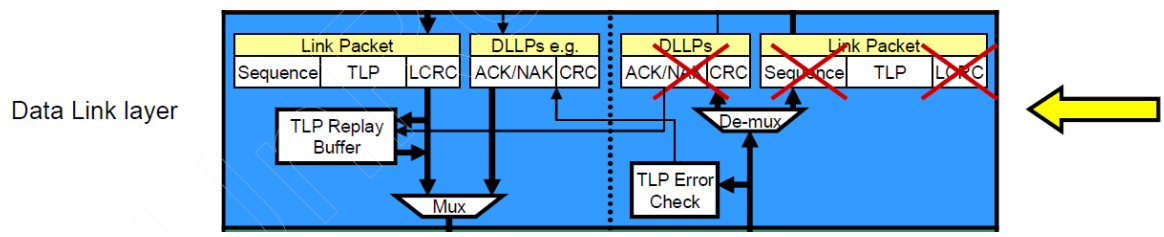


Figure 58: Data Link Layer

4-3 Data Link Layer Packets (DLLP)

The primary responsibility of the PCI Express Data Link Layer is to assure that integrity is maintained when TLPs move between two devices. It also has link initialization and power management responsibilities, including tracking of the link state and passing messages and status between the Transaction Layer above and the Physical Layer below.
In performing its role, the Data Link Layer exchanges traffic with its neighbor using Data Link Layer Packets (DLLPs). DLLPs originate and terminate at the Data Link Layer of each device.

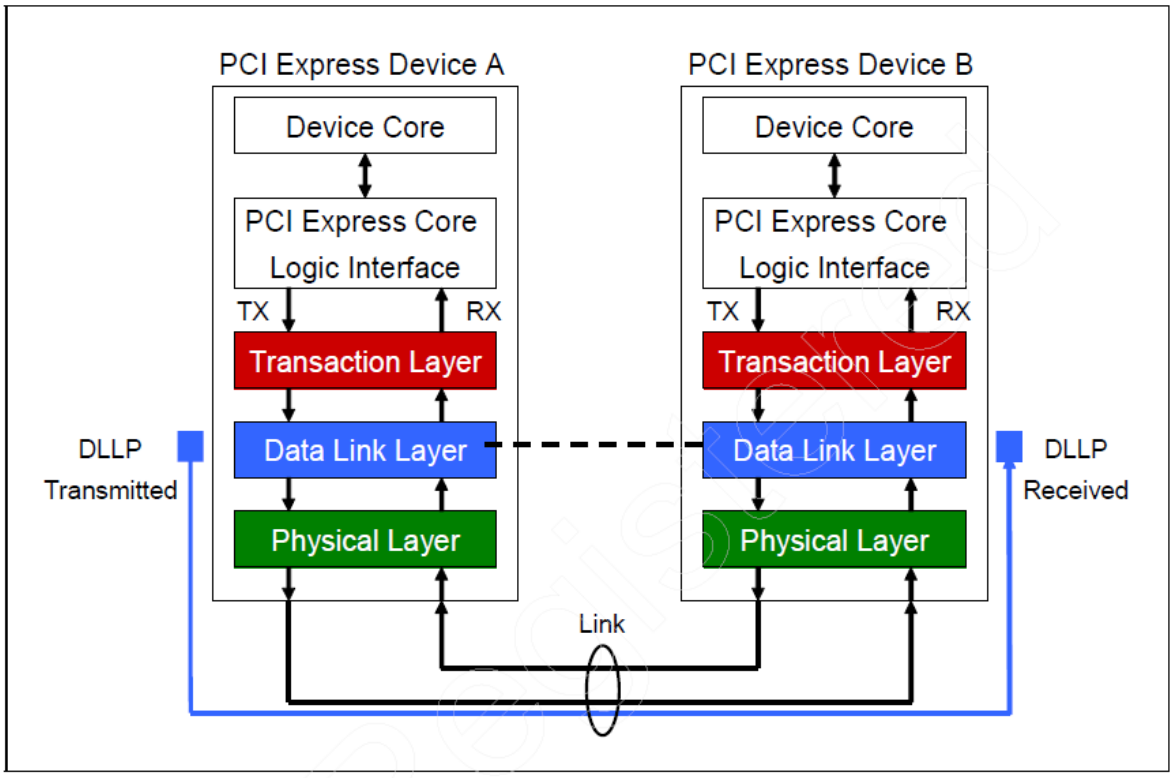


Figure 59: DLLP Flow

There are two important groups of DLLP used in managing a link:

1-TLP Acknowledgment Ack/Nak DLLPs.

2-Flow Control Packet DLLPs

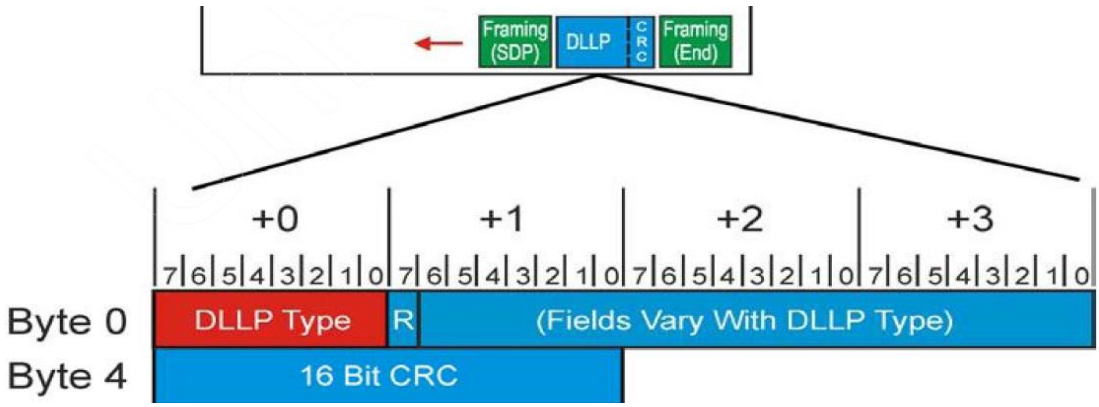


Figure 60: General DLLP Format

4-3-1 TLP Acknowledgment Ack/Nak DLLPs.

Ack DLLP: TLP Sequence number acknowledgement; used to indicate successful receipt of some number of TLPs

Nak DLLP: TLP Sequence number negative acknowledgement; used to initiate a Data Link Layer Retry. As to notify the transmitter that there is error in somewhat packet or error in link and order it to send packets again from Reply Buffer.

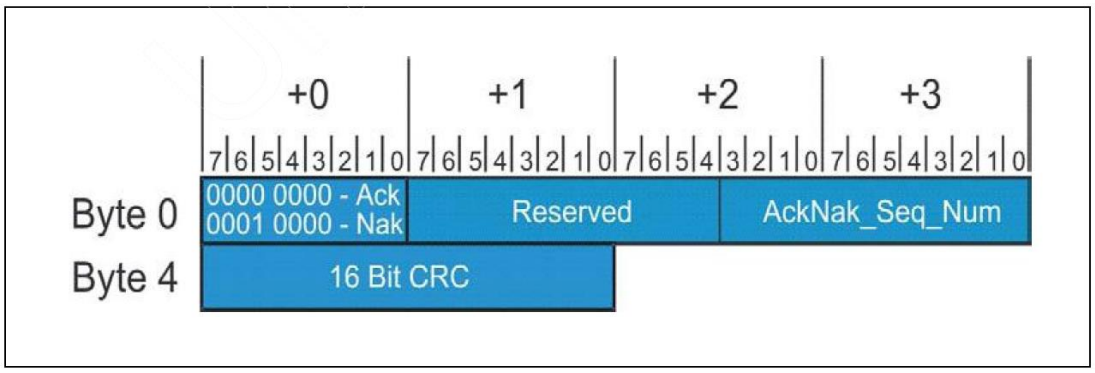


Figure 61: ACK/NAK DLLP Format

4-3-2 Flow Control Packet DLLPs

The device must inform the other device the credits available of specified transaction (e.g posted, non-posted, completions) in its transaction layer to notify another device to make check of its pending TLP. the size of Pending TLP must be less than this Credits so if the size of pending TLP is less than Credits Available in the other device. the transmitter then get permission and send TLP

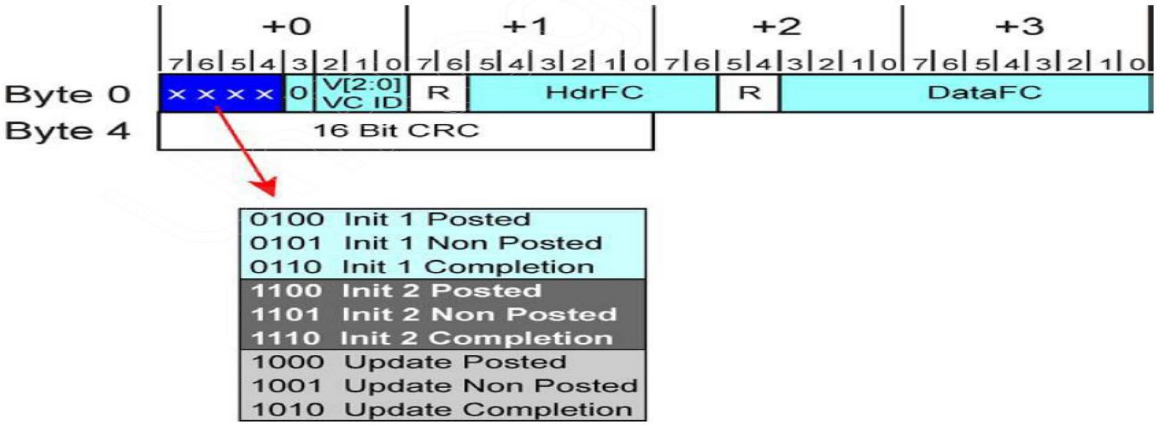


Figure 62: Flow Control DLLP Format

Init 1: means at the beginning of link initialization the device must inform the other device the credits available of specified transaction (e.g posted, non-posted, completions)

Init 2: used as a check for transaction layer in transmitter to notify other layer that the transmitter transaction layer has already known the credits available in other device

Update: used through transferring of packets across link to keep all devices updated with Credits Available in other devices

4-5 ACK/NAK Protocol

The TLP transmission path through the Data Link Layer prepares each TLP for transmission by applying a sequence number, then calculating and appending

a Link CRC (LCRC) which is used to ensure the integrity of TLPs during transmission across a Link from one component to another. TLPs are stored in a retry buffer, and are re-sent unless a positive acknowledgement of receipt is received from the other component. If repeated attempts to transmit a TLP are unsuccessful, the Transmitter will determine that the Link is not operating correctly, and instruct the Physical Layer to retrain the Link If Link retraining fails, the Physical Layer will indicate that the Link is no longer up, causing the Data Link Layer to move to the DL Inactive state

The ACK/NAK protocol associated with the Data Link Layer is described with the aid of Figure 5-2 on page 211 which shows sub-blocks with greater detail. For every TLP that is sent from one device (Device A) to another (Device B) across one Link, the receiver checks for errors in the TLP (using the TLP's LCRC field). The receiver Device B notifies transmitter Device A on good or bad reception of TLPs by returning an ACK or a NAK DLLP. Reception of an ACK DLLP by the transmitter indicates that the receiver has received one or more TLP(s) successfully. Reception of a NAK DLLP by the transmitter indicates that the receiver has received one or more TLP(s) in error. Device A which receives a NAK DLLP then re-sends associated TLP(s) which will hopefully, arrive at the Receiver successfully without error.

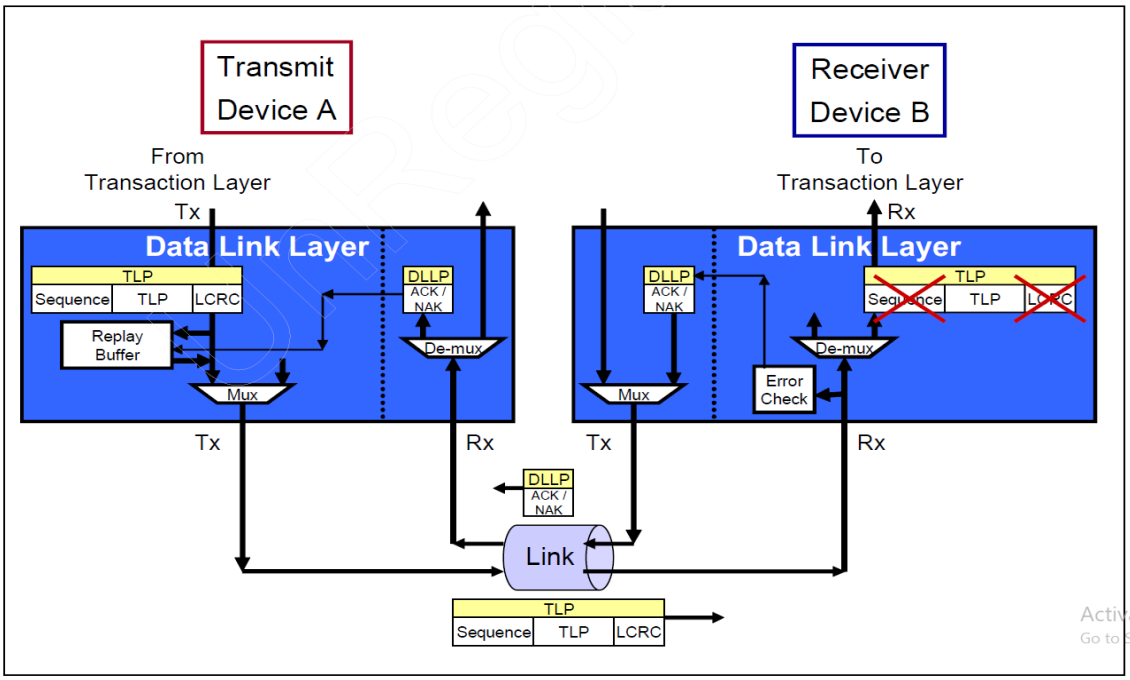


Figure 63: Over View of ACK/NAK Protocol

4-6 Elements of Transmitter ACK/NAK Protocol

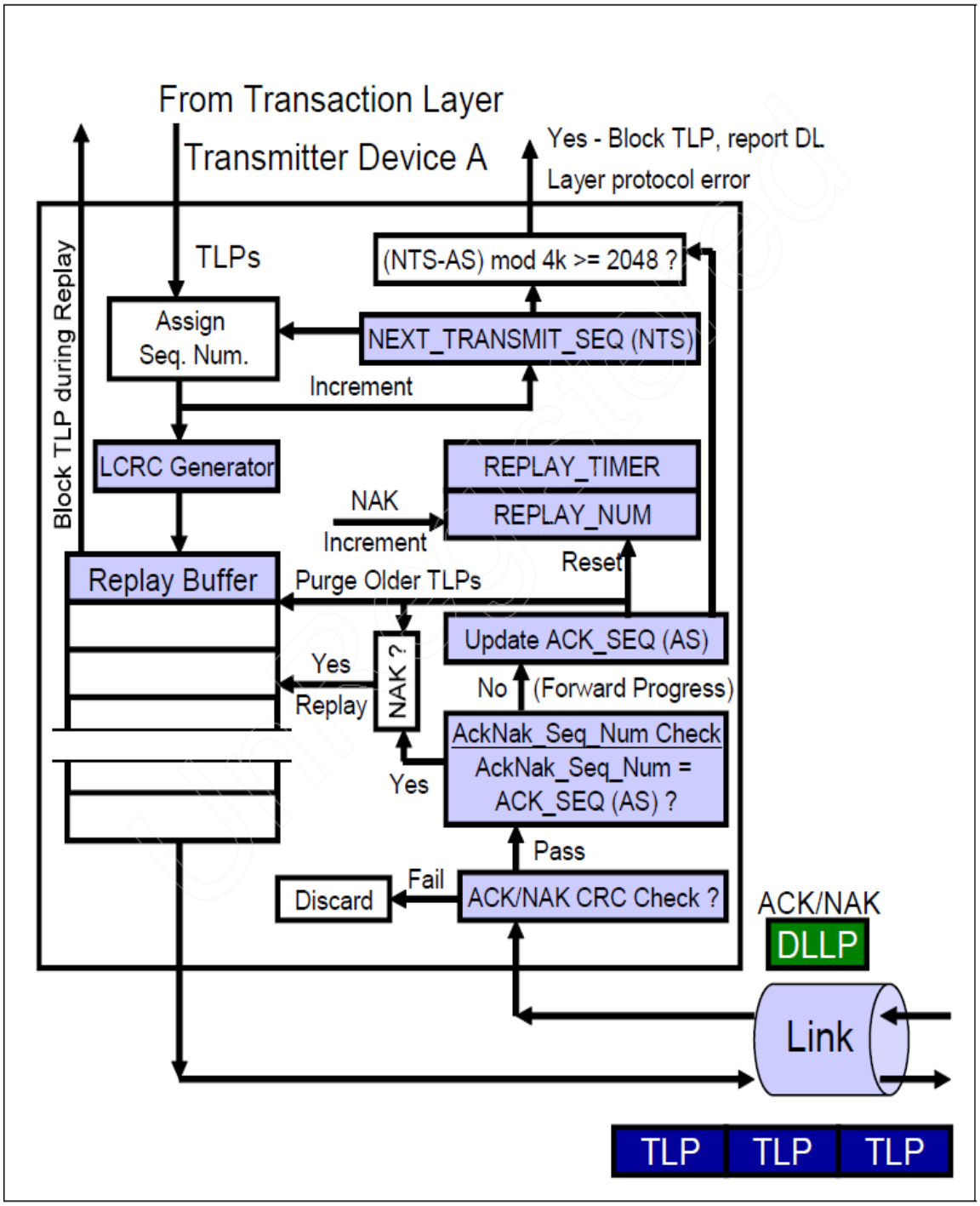


Figure 64: Elements of Transmitter ACK/NAK Protocol

4-6-1 Replay Buffer

The replay buffer stores TLPs with all fields including the Data Link Layer- related Sequence Number and LCRC fields. The TLPs are saved in the order of arrival from the Transaction Layer before transmission. Each TLP in the Replay Buffer contains a Sequence Number which is incrementally greater than the sequence number of the previous TLP in the buffer.

When the transmitter receives acknowledgement via an ACK DLLP that TLPs have reached the receiver successfully, it purges the associated TLPs from the Replay Buffer. If, on the other hand, the transmitter receives a NAK DLLP, it replays (i.e., re-transmits) the contents of the buffer.

4-6-2 Next Transmit Seq Num

This counter generates the Sequence Number assigned to each new transmitted TLP. The counter is a 12-bit counter that is initialized to O at reset, or when the Data Link Layer is in the inactive state. It increments until it reaches 4095 and then rolls over to 0 (i.e., it is a modulo 4096 counter).

4-6-3 LCRC Generator

This module calculates a 32-bit CRC. The LCRC value is added to the tad of the TLP. This module sends the LCRC value in four bytes after the Transaction Layer sends the last byte of the TLP.

4-6-4 Replay Timer

REPLAY_TIMER - Counts time since last Ack or Nak DLLP received

The Timer is reset to zero and restart when

1. A replay event occurs and the last symbol of TLP is replayed.
2. For ACK DLLP received as long as there is unacknowledged TLP in replay buffer

The Timer is reset to Zero and held when

1. There are no TLPs to transmit, or when the Replay Buffer is empty.
2. A NAK DLLP is received. The timer restarts when replay begins.
3. When the timer expires.
4. The Data Link Layer is inactive.

Timer is Held during Link training or re-training.

When this Timer expires the Data Link layer of transmitter know that there is problem in link so it sends all packets in replay buffer.

The timer is loaded with a value that reflects the worst-case latency for the return of an ACK or NAK DLLP. This time depends on the maximum data payload allowed for a TLP and the width of the Link.

The equation to calculate the REPLAY_TIMER value required is:

$$\left(\frac{(Max\_Payload\_Size + TLPOverhead) * AckFactor}{LinkWidth} + InternalDelay\right) * 3 + Rx\_L0s\_Adjustment$$

Figure 65: Equation of Replay Timer

Where

Max Payload_Size: is the value in the Max_Payload_Size field of the Device Control register. For a multi-function device whose Max_Payload_Size settings are identical across all functions, the common Max_Payload_Size setting must be used. For a multi-function

Device whose Max_Payload_Size settings are not identical across all functions, the selected Max_Payload_Size setting is implementation specific, but it's recommended to use the largest Max_Payload_Size setting across all functions.

TLP Overhead: represents the additional TLP components which consume Link

bandwidth (header, LCRC, framing Symbols) and is treated here asa constant value of 28 Symbols

AckFactor : represents the number of maximum size TLPs which can be received before an Ack is sent, and is used to balance Link bandwidth efficiency and retry buffer size – the value varies according to Max_Payload_Size and Link width

LinkWidth: is the operating width of the Link

InternalDelay : represents the internal processing delays for received TLPs and transmitted DLLPs, and is treated here as a constant value of 19 Symbol Times

Rx_L0s_Adjustment: equals the time required by the components receive circuits to exit from L0s to L0 (as to receive an Ack DLLP from the other component on the Link)

4-6-5 Replay_Num Count

This 2-bit counter stores the number of replay attempts following either reception of a NAK DLLP, or a REPLAY_TIMER time-out. When the REPLAY_NUM count rolls over from 11b to 00b, the Data Link Layer triggers a Physical Layer that the link need retrained and data link layer enter inactive state and after that It waits for completion of re-training before attempting to transmit TLPs once again. The REPLAY_NUM counter is initialized to 00b at reset, or when the Data Link Layer is inactive. It is also reset whenever an ACK is received, indicating that forward progress is being made in transmitting TLPs.

4-6-6 ACKD_SEQ Count

This 12-bit register tracks or stores the Sequence Number of the most recently received ACK or NAK DLLP. It is initialized to all is at reset, or when the Data Link Layer is inactive. This register is updated with the AckNak_Seq Num [11:0] field of a received ACK or NAK DLLP. The ACKD_SEQ count is compared with the NEXT_TRANSMIT_SEQ count.

4-6-7 DLLP CRC Check

This block checks for CRC errors in DLLPs returned from the receiver. Good DLLPs are further processed. If a DLLP CRC error is detected, the DLLP is discarded and an error reported. No further action is taken.

4-7 Elements of Receiver ACK/NAK Protocol

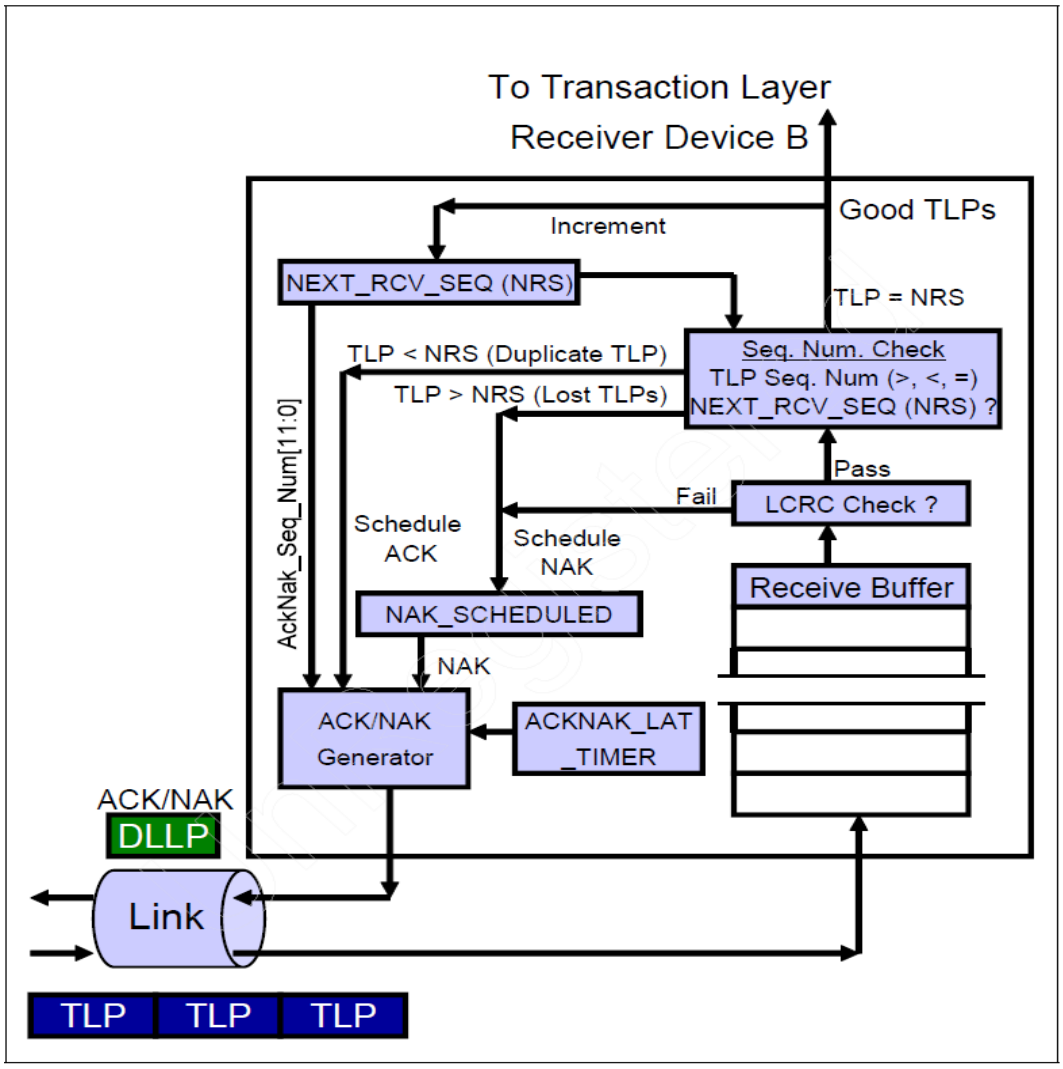


Figure 66: Elements of Receiver ACK/NAK Protocol

4-7-1 Receive Buffer

The receive buffer temporarily stores received TLPs while TLP CRC and Sequence Number checks are performed. If there are no errors, the TLP is processed and transferred to the receiver ‘s Transaction Layer. If there are errors associated with the TLP, it is discarded and a NAK DLLP may be scheduled. If the TLP is a duplicate TLP it is discarded and an ACK DLLP is scheduled. If the TLP is a ‘nullified’ TLP, it is discarded and no further action is taken

4-7-2 LCRC Check

This block checks for LCRC errors in the received TLP using the TLP’s 32-bit LCRC fie1d.

4-7-3 Next_RCV_SEQ_Count

The 12-bit NEXT_RCV_SEQ counter keeps track of the next expected TLP’s Sequence Number. This counter is initialized to 0 at reset, or when the Data Link Layer is inactive. This counter is incremented once for each good TLP received that is forwarded to the Transaction Layer. The counter rolls over to 0 after reaching a value of 4095. The counter is not incremented for TLPs received with CRC error, nullified TLPs, or TLPs with an incorrect Sequence Number.

4-7-4 Sequence Number Check

• If the TLP’s Sequence Number = NEXT_RCV_SEQ count, the TLP is accepted, processed and forwarded to the Transaction Layer. NEXT_RCV_SEQ count is incremented. The receiver continues to process inbound TLPs and does not have to return an ACK DLLP until the ACKNAK_LATENCY_TIMER expires or exceeds its set value.

• If the TLP’s Sequence Number is an earlier Sequence Number than NEXT_RCV_SEQ count and with a separation of no more than 2048 from NEXT_RCV_SEQ count, the TLP is a duplicate TLP. It is discarded and an ACK DLLP is scheduled for return to the transmitter.

•If the TLP’s Sequence Number is a later Sequence Number than NEXT_RCV_SEQ count, or for any other case other than the above two conditions, the TLP is discarded and a NAK DLLP may be scheduled (more on this later) for return to the transmitter.

4-7-5 Nak scheduled flag

The NAK_SCHEDULED flag is set when the receiver schedules a NAK DLLP to return to the remote transmitter. It is cleared when the receiver sees the first TLP associated with the replay of a previously-Nak’d TLP. The receiver shouldn’t schedule additional NAK DLLP forbad TIPs received while the NAK_SCHEDULED flag is set.

4-7-6 ACKNAK_Latency_Timer

The ACKNAK_LATENCY_TIMER monitors the elapsed time since the last ACK or NAK DLLP was scheduled to be returned to the remote transmitter. The receiver uses this timer to ensure that it processes TLPs promptly and returns an ACK or a NAK DLLP when the timer expires or exceeds its set value.

The timer resets to 0 and holds when:

1. . All received TLPs have been acknowledged.
2. . The Data Link Layer is in the inactive state.

When the timer expires the receiver schedules and Ack DLLP of last unacknowledged good TLP

The receiver's ACKNAK_LATENCY_TIMER is loaded with a value that reflects the worst-case transmission latency in sending an ACK or NAK in response to a received TLP.

This time depends on the anticipated maximum payload size and the widthof the Link.

The equation to calculate the ACKNAK_LATENCY_TIMER value required

$$\frac{(Max\_Payload\_Size + TLPOverhead) * AckFactor}{LinkWidth} + InternalDelay + Tx\_L0s\_Adjustment$$

Figure 67: ACKNAK_LATENCY_TIMER equation

Max_Payload_Size: is the value in the Max_Payload_Size field of the Device Control register. For a multi-function device whose Max_Payload_Size settings are identical across all functions, the common Max_Payload_Size setting must be used. For a multi-function device whose Max_Payload_Size settings are not identical across all functions, the selected Max_Payload_Size setting is implementation specific, but it's recommended to use the smallest Max_Payload_Size setting across all functions.

TLP Overhead: represents the additional TLP components which consume Link bandwidth (header, LCRC, framing Symbols) and is treated here as a constant value of 28 Symbols.
30 AckFactor represents the number of maximum size TLPs which can be received before an Ack is sent, and is used to balance Link bandwidth efficiency and retry buffer size – the value varies according to Max_Payload_Size and Link width.

LinkWidth: is the operating width of the Link.

InternalDelay : represents the internal processing delays for received TLPs and transmitted DLLPs, and is treated here as a constant value of 19 Symbol Times.

Tx_L0s_Adjustment: if L0s is enabled, the time required for the Transmitter to exit L0s

4-7-7 ACK/NAK DLLP Generator

This block generates the ACK or NAK DLLP upon command from the LCRC or Sequence Number check block. The ACK or NAK DLLP contains an AckNak_Seq_Num[11:0] field obtained from NEXT_RCV_SEQ counter . Ack or NAK DLLPs contain AckNak_Seq_Num [11:0] value equal to NEXT_RCV_SEQ – 1

4-8 Scheduling an ACK DLLP

If the receiver does not detect an LCRC error or a Sequence Number related error associated with a received TLP, it accepts the TLP and sends it to the Transaction Layer. The NEXT_RCV_SEQ counter is incremented and the receiver is ready for the next TLP. At this point, the receiver can schedule an ACK DLLP with the Sequence Number of the received TLP.

Alternatively, the receiver could also wait for additional TLPs and schedule an ACK DLLP with the Sequence Number of the last good TLP received. The receiver is allowed to accumulate a number of good TLPs and then sends one aggregate ACK DLLP with a Sequence Number of the latest good TLP received. The coalesced ACK DLLP acknowledges the good receipt of a collection of TLPs starting with the oldest TLP in the transmitter's Replay Buffer and ending with the TLP being acknowledged by the current ACK DLLP. By doing so, the receiver optimizes the use of Link bandwidth due to reduced ACK DLLP traffic. .When the ACKNAK_LATENCY_ TIMER expires or exceeds its set value and TLPs are received, an ACK DLLP with a Sequence Number of the last good TLP is returned to the transmitter.

4-9 Scheduling a NAK DLLP

Upon receipt of a TLP, the first type of error condition the receiver may detect is a TLP LCRC error) The receiver discards the had TLP. If the NAK_SCHEDULED flag is clear, it schedules a NAK DLLP to return to the transmitter. The NAK_SCHEDULED flag is then set. The receiver uses the NEXT_RCV_SEQ count - 1 count value as the AckNak_Seq_Num [11:0] field in the NAK DLLP At the time the receiver schedules a NAK DLLP to return to the transmitter, the Link may be in use to transmit other queued TLPs, DLLPs or PLPs. In that case, the receiver delays the transmission of the NAK DLLP . When the Link becomes available, however, it sends the NAK DLLP to the remote transmitter. The transmitter replays the TLPs from the Replay Buffer.

In the meantime, TLPs currently en route continue to arrive at the receiver. These TLPs have later Sequence Numbers than the NEXT_RC V_SEQ count. The receiver discards them. The specification is unclear about whether the receiver should schedule a NAK DLLP for these TLPs. It is the authors' interpretation that the receiver must not schedule the return of additional NAK DLLPs for subsequently received TLPs while the NAK_SCHEDULED flag remains set.

The receiver detects a replayed TLP when it receives a TLP with Sequence Numbers that matches NEXT_RC V_SEQ count. If the replayed TLPs arrive with no errors, the receiver increments NEXT_RCV_SEQ count and clears the NAK_SCHEDULED flag. The receiver may schedule an ACK DLLP for return to the transmitter if the ACKNAK_LATENCY_TIMER expires. The good replayed TLPs are forwarded to the Transaction Layer.

4-10 Design and implementation (Block Diagram)

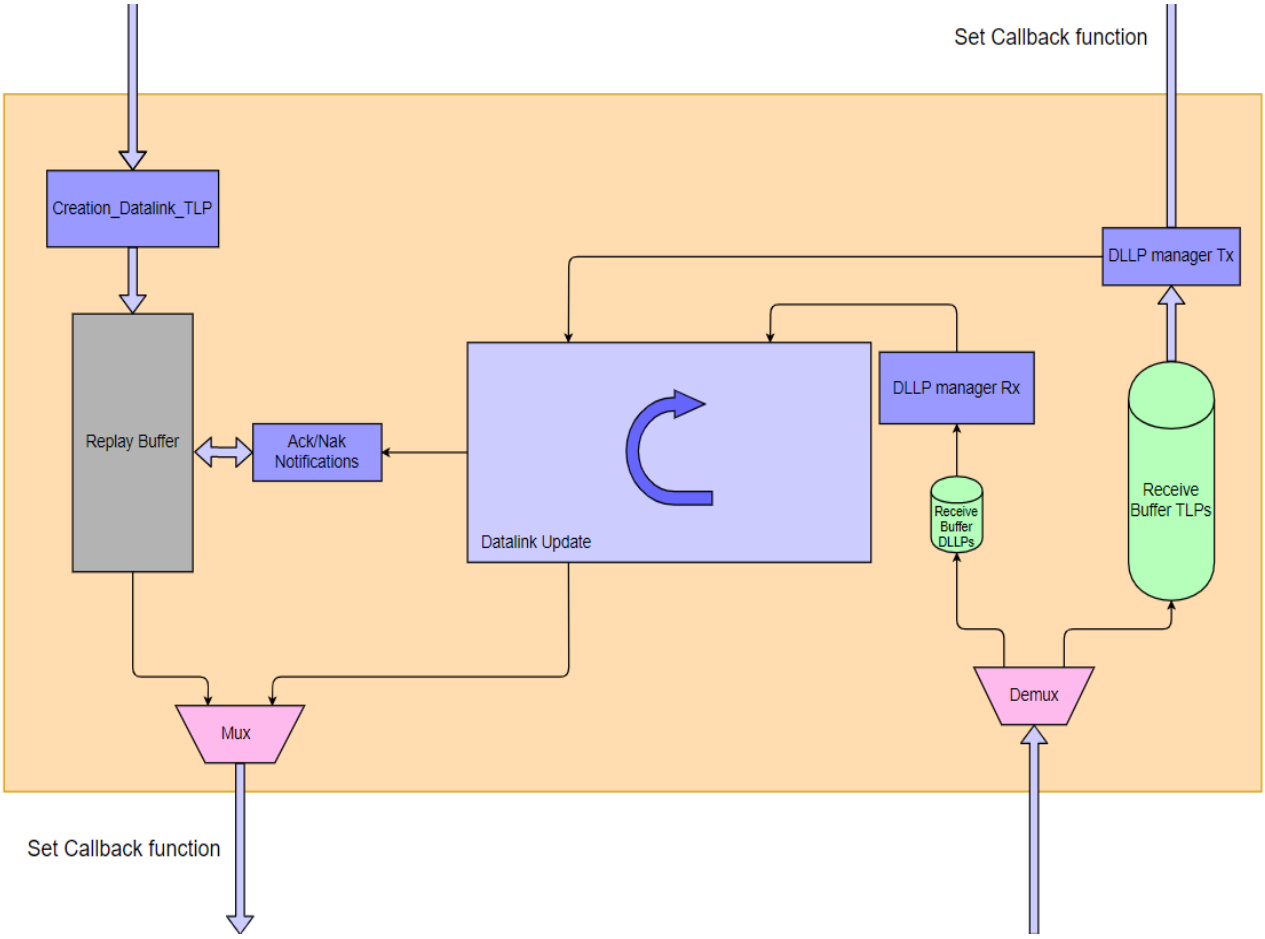


Figure 68: Design and Implementation of Data Link Layer

4-11 Block Description

4-11-1 Creation_DataLink_TLP

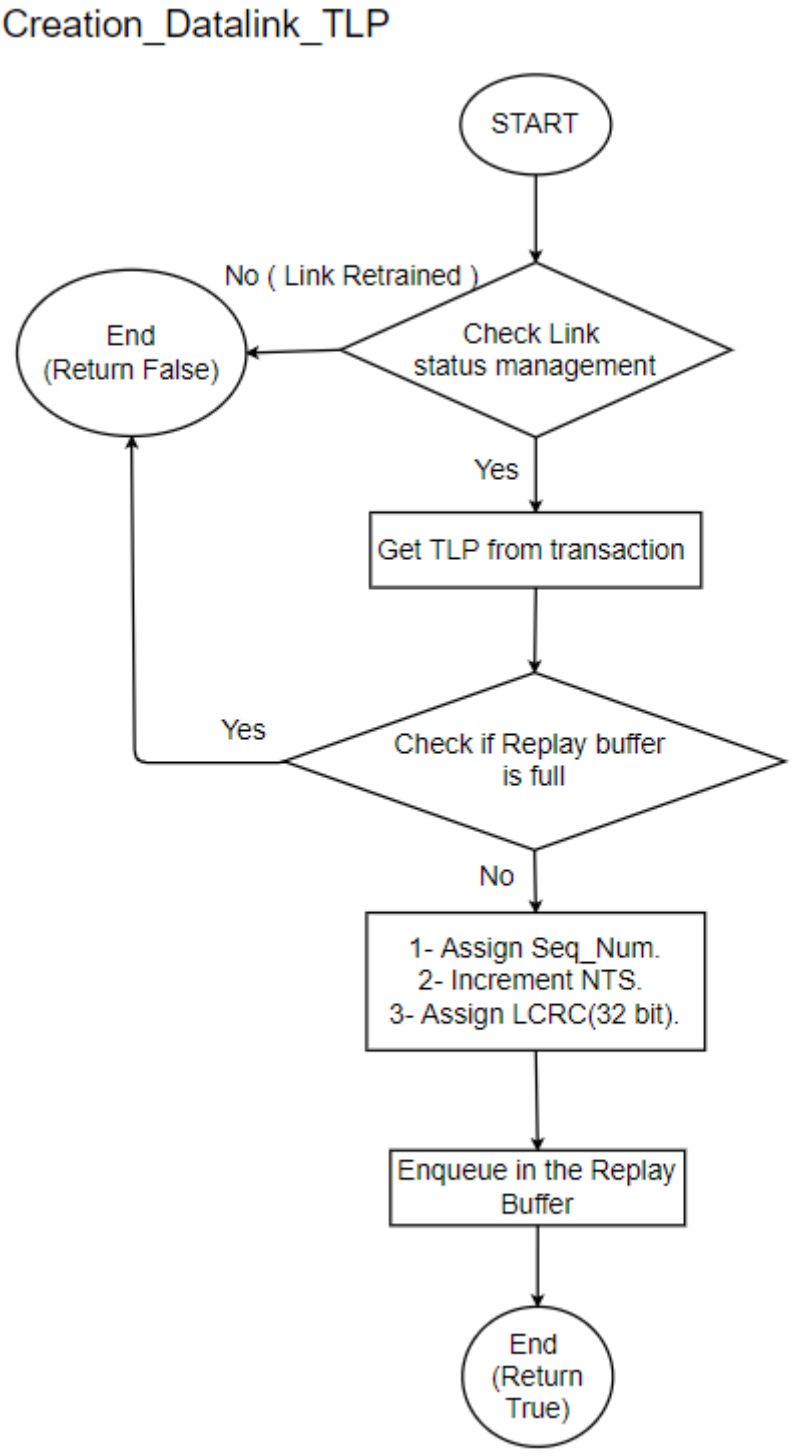


Figure 69: flow chart of Creation_Datalink_TLP

4-11-2 DLLP Manager Rx

Manages the reception of the DLLPs. If an Ack or Nak DLLP has been received, then the DLLP Manager Rx passes the sequence number of the TLP acknowledged or unacknowledged to the Ack_Nak_Notifications.

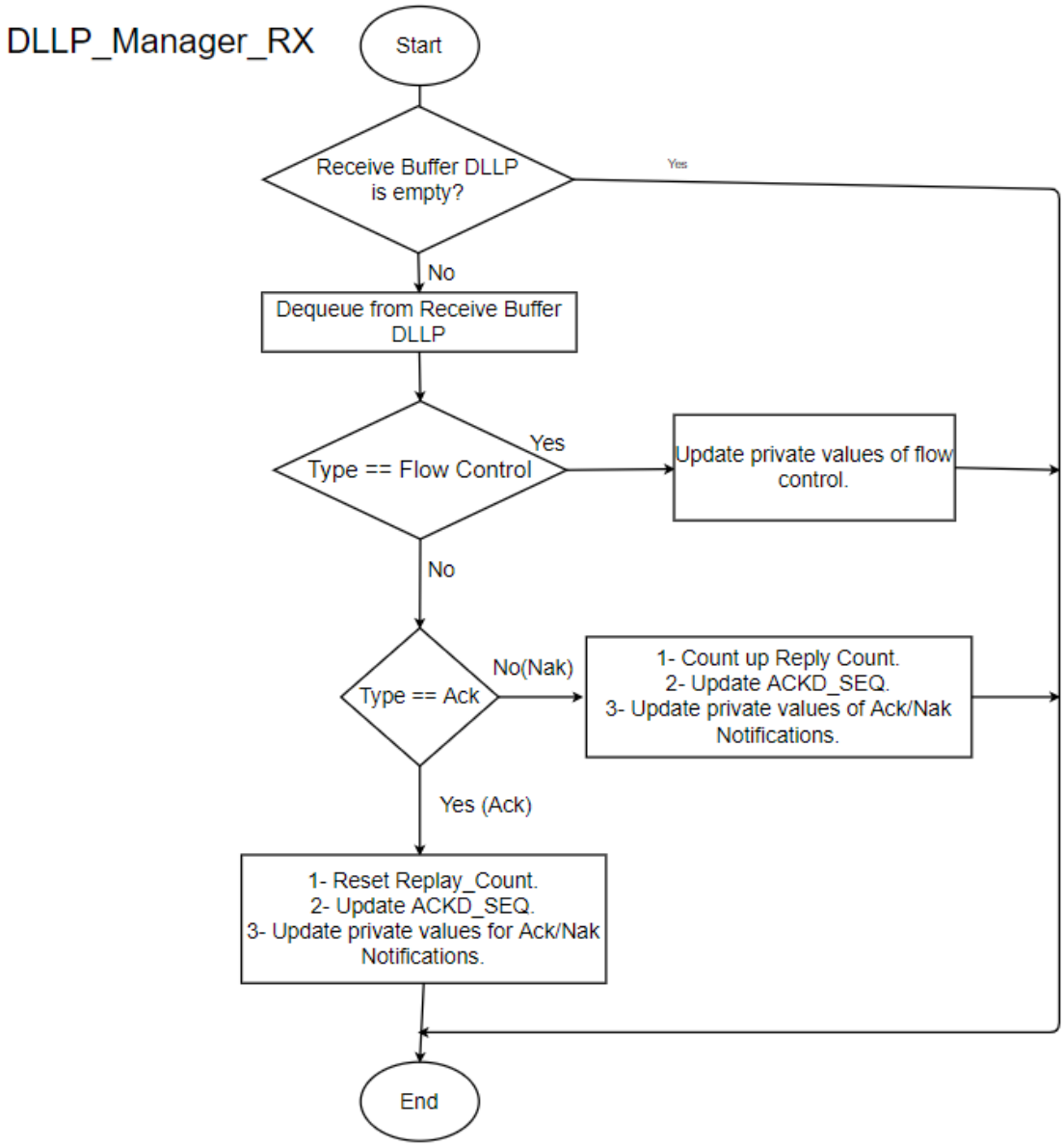


Figure 70: Flow chart of DLLP Manager Rx

4-11-3 DLLP Manager Tx

The DLLP Manager Tx module forms DLLPs,Ack for Good TLP and Nak for Bad TLP,and then schedules and transmits a DLLP to the other component on the Link

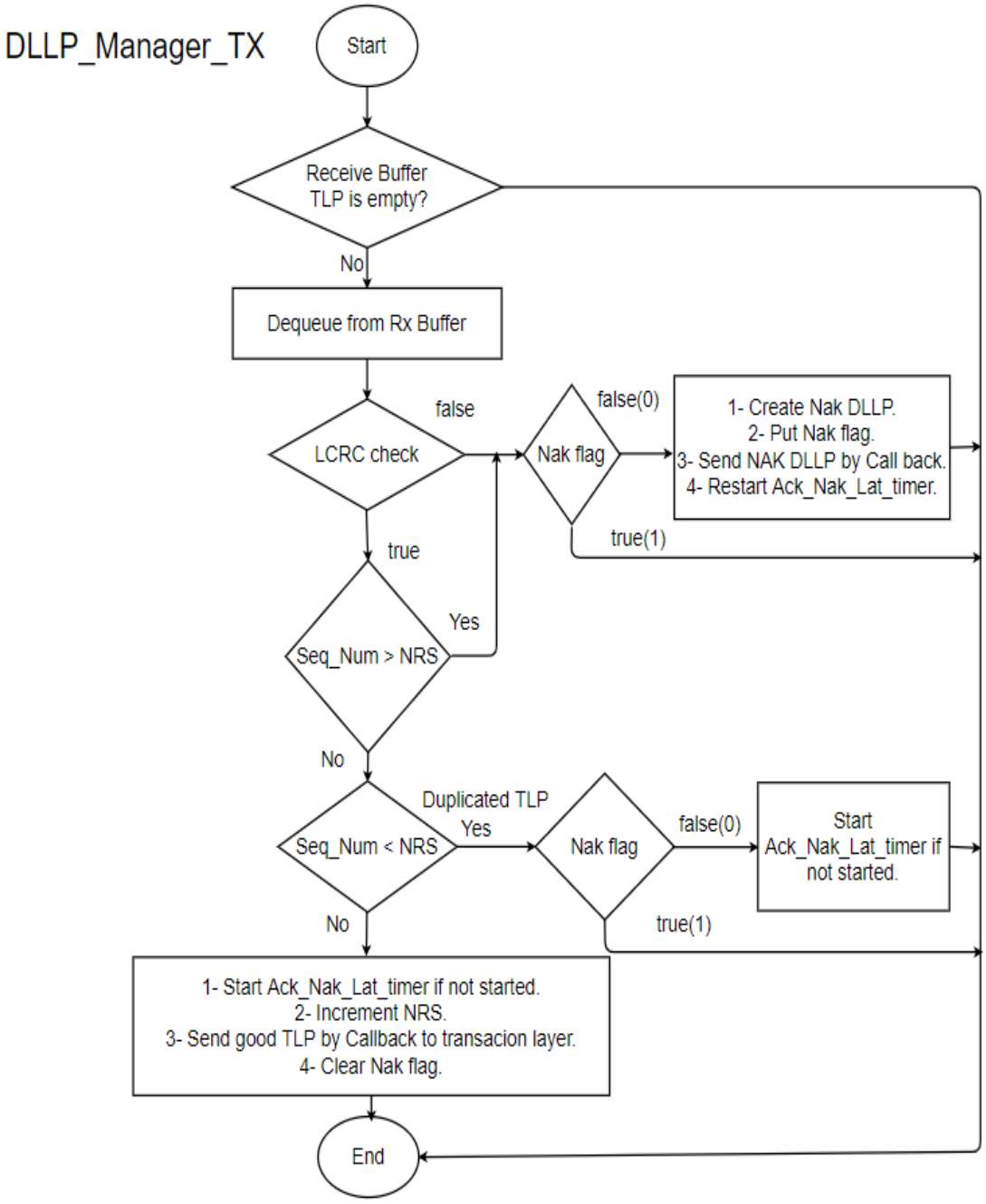


Figure 71: Flow chart of DLLP Manager Tx

4-11-4 Ack_Nak_Notifications

This module notifies to the Replay Buffer the TLPs acknowledged or unacknowledged in the Receiver.

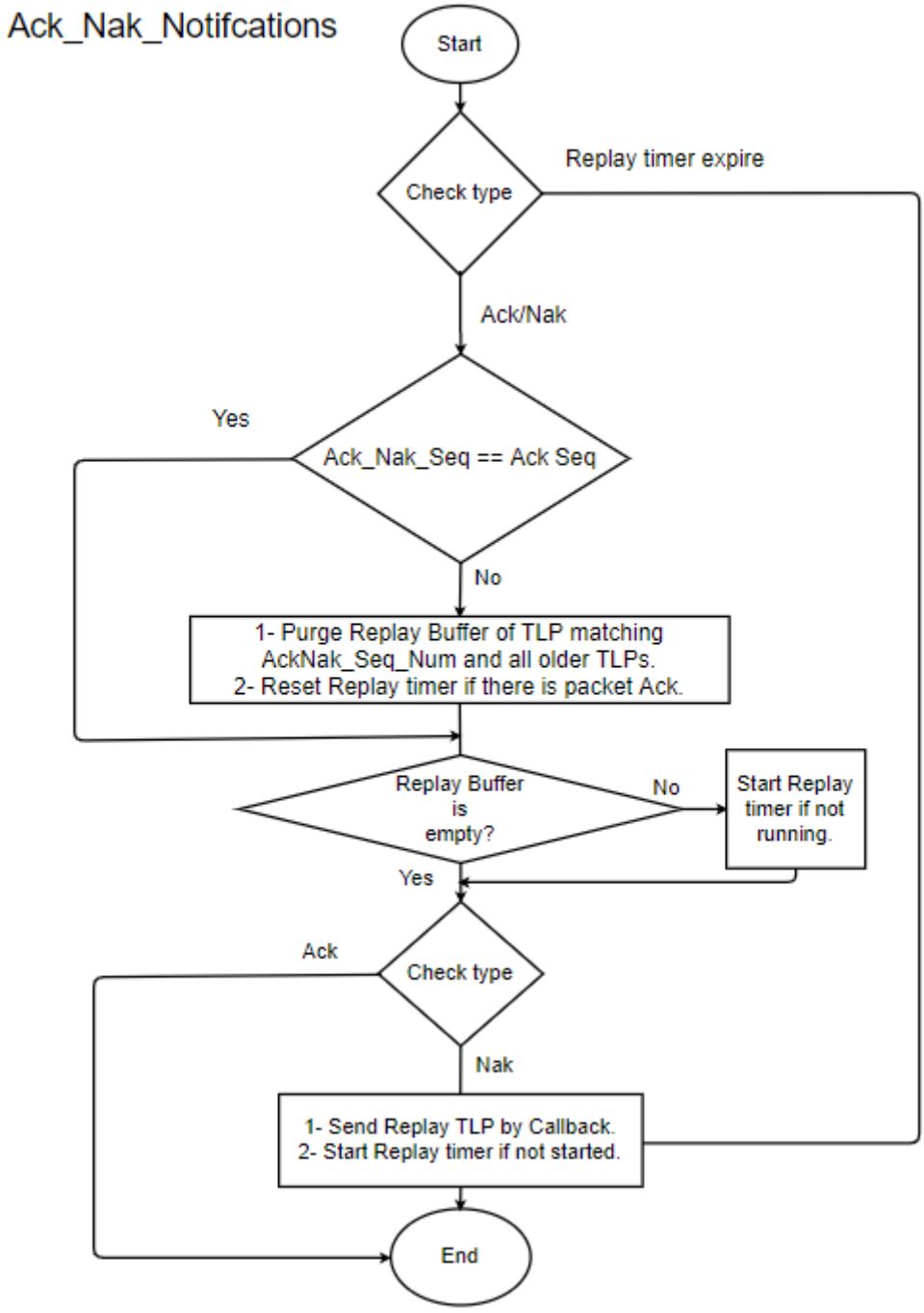


Figure 72: Flow chart of Ack_Nak_Notifcations

4-11-5 Data_link_Update

It is the controller of Data Link, work as thread, it calls All blocks in Data Link by polling and check for expiration of timers and take action in case of expirations of one of these timers and also this block sends new packets to other layer

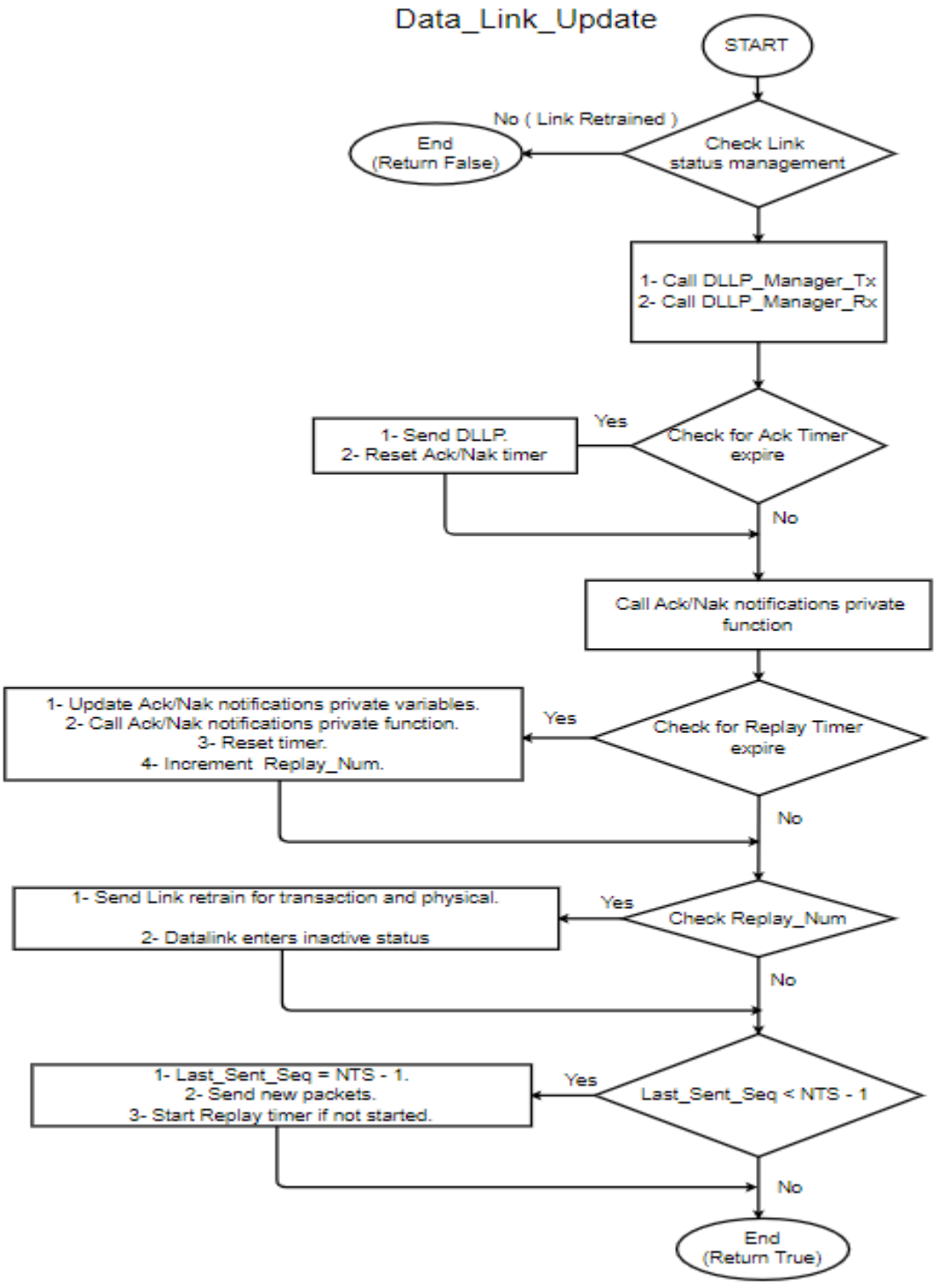


Figure 73: Flow chart of Data_Link_Update

4-12 Interfacing with Transaction Layer

In Our Design We Designed Transaction Layer as a class and Data Link Layer as another class and Class Transaction layer can interface Data Link Class through four Interfaces(APIs) and Class Data Link Layer Can Interface Transaction Layer Class through One Interface(API)

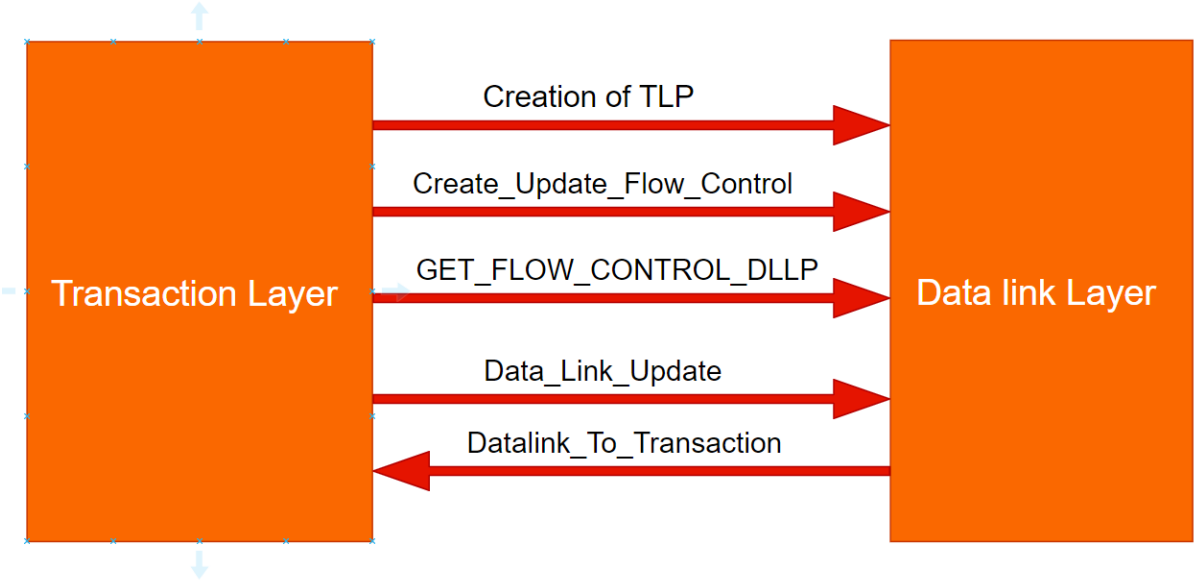


Figure 74: Interfacing Transaction Layer with Data Link Layer and vice versa

4-12-1 Creation of TLP

This API first check link status management if the link is active it gets TLP from Transaction and Assign Sequence Number and LCRC and then put the packet in Reply Buffer

4-12-2 Create_Update_Flow_Control

As we said before, Transaction Layer should notify other devices with the credits available in their Transaction Layer according to each type of Transaction this can be done by transmitting Credits Alloc for both Header and Data and type of Transaction to Data Link Layer to Create Flow Control Packet and send it to other device, this can be done by this Create_Update_Flow_Control API

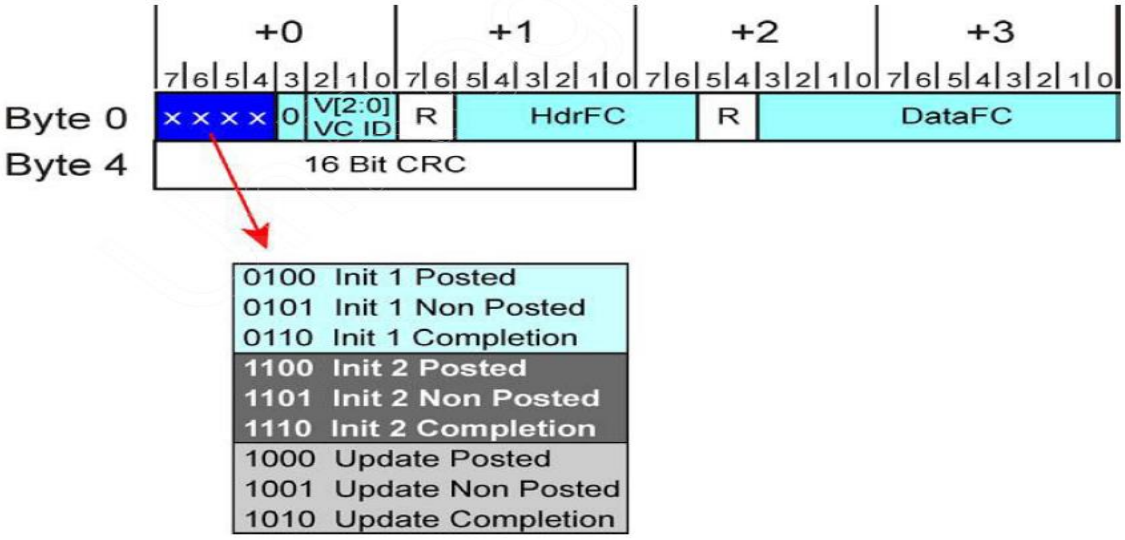


Figure 75: Types and Format of Flow Control Packets

4-12-3 GET_FLOW_CONTROL_DLLP

As we said before DLLP terminates at Data Link Layer, so when the device send flow control DLLP carrying Credits Alloc to other device. the Transaction Layer of receiving device want to know the Credits Alloc in other Layer to update its Credit Limit Register for both data and header and know if it can send its pending TLP as the size of Pending TLP must be less than Credits Limit, this can be done by this API, this API returns the value of Credits Alloc of other Transaction Layer

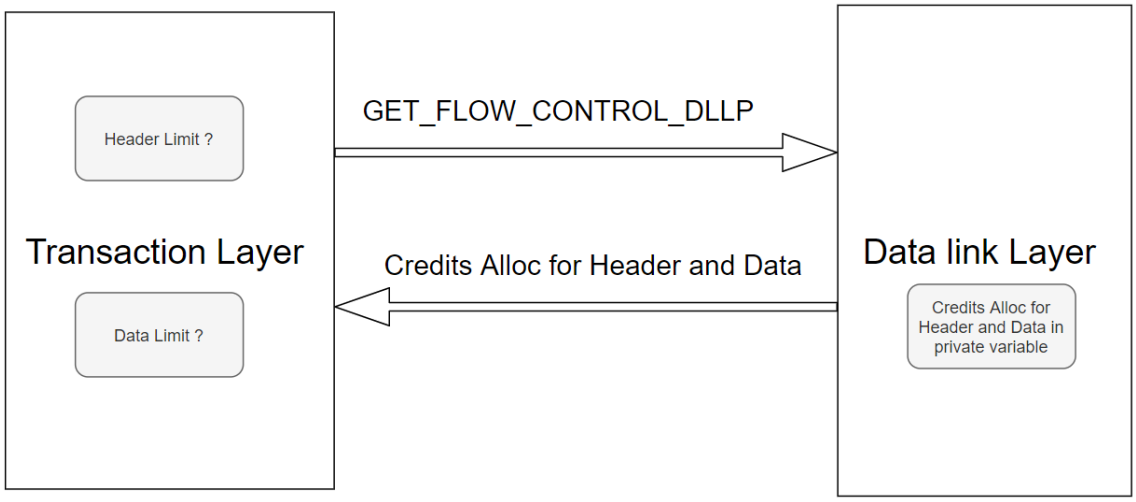


Figure 76: illustration of GET_FLOW_CONTROL_DLLP

4-12-4 Data_Link_Update

The thread of transaction Layer should call this API because it is the Controller of Data link that calls all blocks in Data Link and managing transferring packets and DLLP through Data Link

The mechanism of Data_Link_Update when it is called by the thread of transaction layer is as shown in its Block diagram

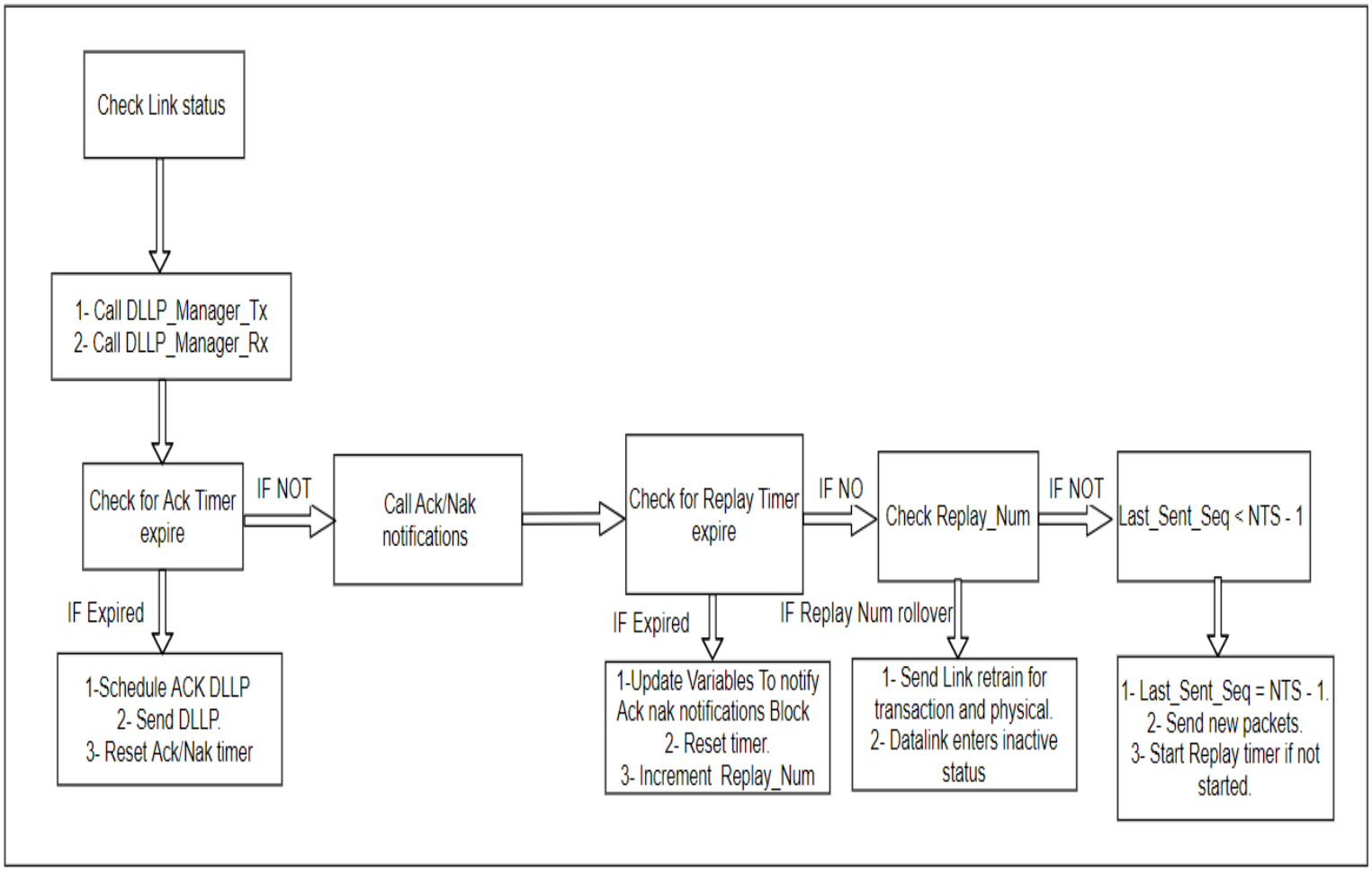


Figure 77: Block Diagram Data_Link_Update

4-12-5 Data_Link_To_Transaction

Before the Data Link Calls this API, Data link must Decompose Packet as to remove Sequence number and LCRC, this can be done by DLLP Manger Tx Block, After Decomposing it class Data Link Call this API, when calling this API, the TLP is passed from Data Link Layer to Transaction Layer and enqueued in Rx Buffer according to the type of transaction of this TLP (e.g posted transaction enqueued)

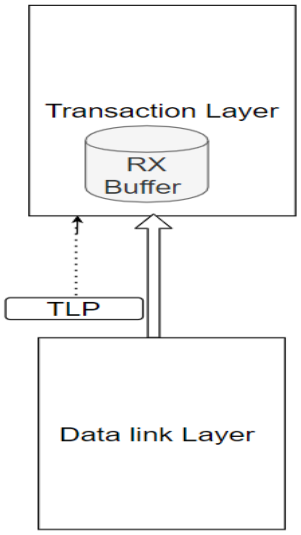


Figure 78: Illustration Data_Link_To_Transaction

4-13 Recommended Priority to Schedule Packets

A device may have many types of TLPs, and DLLPs to transmit on a given Link. The following is a recommended but not required set of priorities for scheduling packets:

1. Completion of any TLP or DLLP currently in progress (highest priority).
2. PLP transmissions.
3. NAK DLL-P.
4. ACK DLLP.
5. FC (Flow Control) DLLP.
6. Replay Buffer re-transmissions.
7. TLPs that are waiting in the Transaction Layer.
8. All other DLLP transmissions (lowest priority)

Chapter 5

Physical Layer

5-1 Physical layer overview

The Physical Layer **shown in** connects to the Link on one side and interfaces to the Data Link Layer on the other side. The Physical Layer processes outbound packets before transmission to the Link and processes inbound packets received from the Link. The two sections of the Physical Layer associated with transmission and reception of packets are referred to as the transmit logic and the receive logic throughout this chapter.

The transmit logic of the Physical Layer essentially processes packets arriving from the Data Link Layer, then converts them into a serial bit stream. The bit stream is clocked out at 2.5 Gbits/s/Lane onto the Link.

The receive logic clocks in a serial bit stream arriving on the Lanes of the Link with a clock that is recovered from the incoming bit stream. The receive logic converts the serial bit steam into a parallel symbol stream, processes the incoming symbols, assembles packets and sends them to the Data Link Layer.

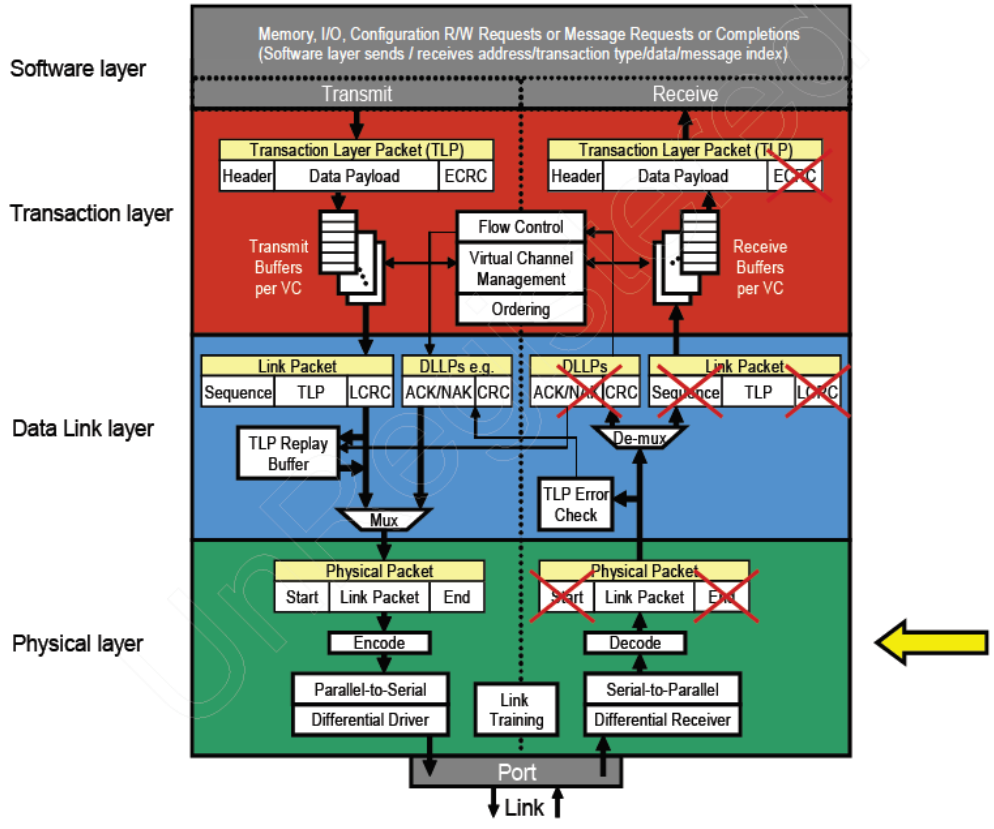


Figure 79: The Physical layer

In the future, data rates per Lane are expected to go to 5 Gbits/s, 10 Gbits/s and beyond. When this happens, an existing design can be adapted to the higher data rates by redesigning the Physical Layer while maximizing reuse of the Data Link Layer, Transaction Layer and Device Core/Software Layer. The Physical Layer may be designed as a standalone entity separate from the Data Link Layer and Transaction Layer. This allows a design to be migrated to higher data rates or even to an optical implementation if such a Physical Layer is supported in the future.

Two sub-blocks make up the Physical Layer. These are the logical Physical Layer and the electrical Physical Layer as shown below. This chapter describes the logical sub-block, and the next chapter describes the electrical sub-block. Both sub-blocks are split into transmit logic and receive logic (independent of each other) which allow dual simplex communication.

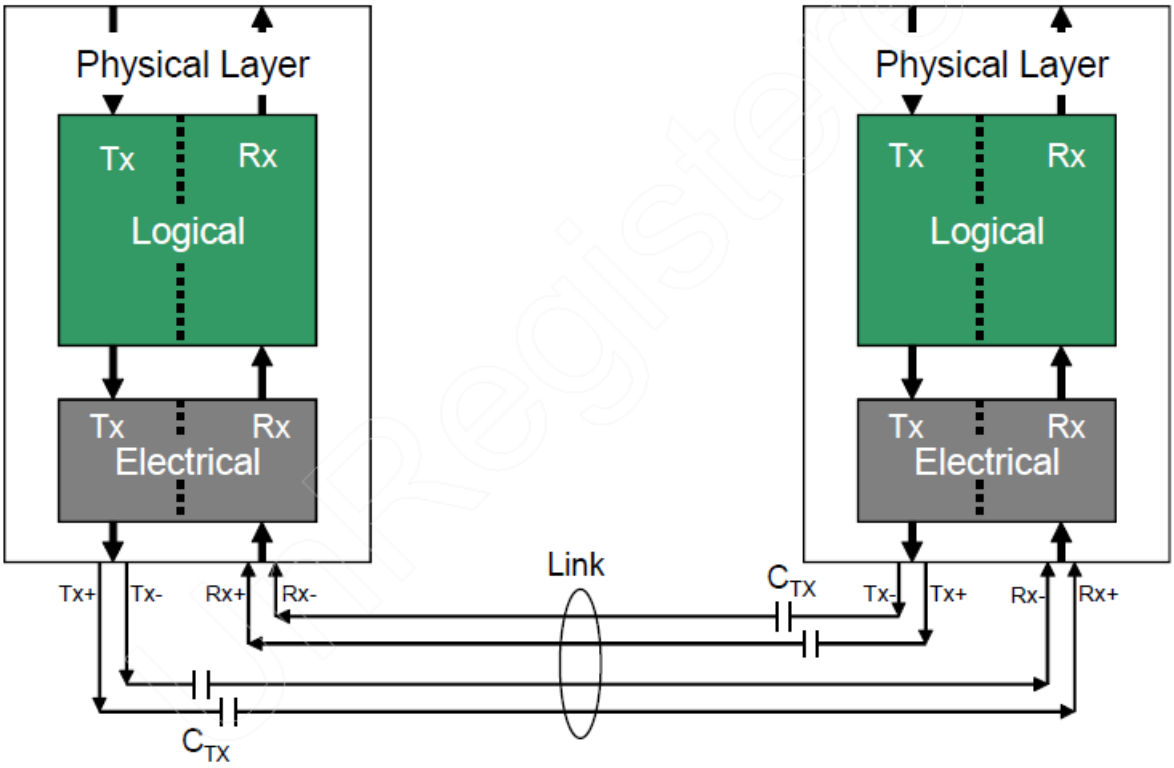


Figure 80: The Electrical and the Logical Physical

5-2 Transmitter Logic

Figure 81 shows the elements that make up the transmit logic:

- Multiplexer (mux).
- Byte striping logic (only necessary if the link implements more than one data lane).
- Scramblers.
- 8b/10b encoders.
- Parallel-To-Serial converters.

TLPs and DLLPs from the Data Link layer are clocked into a Tx Buffer. With the aid of a multiplexer, the Physical Layer frames the TLPs or DLLPs with Start and End characters. These characters are framing symbols which the receiver device uses to detect start and end of packet.

The framed packet is sent to the Byte Striping logic which multiplexes the bytes of the packet onto the Lanes. One byte of the packet is transferred on one Lane, the next byte on the next Lane and so on for the available Lanes.

The Scrambler uses an algorithm to pseudo-randomly scramble each byte of the packet. The Start and End framing bytes are not scrambled. Scrambling eliminates repetitive patterns in the bit stream. Repetitive patterns result in large amounts of energy concentrated in discrete frequencies which leads to significant EMI noise generation. Scrambling spreads energy over a frequency range, hence minimizing average EMI noise generated.

The scrambled 8-bit characters (8b characters) are encoded into 10-bit symbols (10b symbols) by the 8b/10b Encoder logic. And yes, there is a 25% loss in trans-mission performance due to the expansion of each byte into a 10-bit character. A Character is defined as the 8-bit un-encoded byte of a packet. A Symbol is defined as the 10-bit encoded equivalent of the 8-bit character. The purpose of 8b/10b Encoding the packet characters is primarily to create sufficient 1-to-0 and 0-to-1 transition density in the bit stream so that the receiver can re-create a receive clock with the aid of a receiver Phase Lock Loop (PLL). Note that the clock used to clock the serial data bit stream out of the transmitter is not itself transmitted onto the wire. Rather, the receive clock is used to clock in an inbound packet.

The 10b symbols are converted to a serial bit stream by the Parallel-to-Serial converter This logic uses a 2.5 GHz clock to serially clock the packets out on each Lane. The serial bit stream is sent to the electrical sub-block which differentially transmits the packet onto each Lane of the Link.

5-3 Receive Logic

Figure 81 shows the elements that make up the receive logic:

- Receive PLL.
- Serial-to-parallel converter.
- Elastic buffer.
- 8b/10b decoder.
- De-scrambler.
- Byte un-striping logic (only necessary if the link implements more than one data lane).
- Control character removal circuit.
- Packet receive buffer.

As the data bit stream is received, the receiver PLL is synchronized to the clock frequency with which the packet was clocked out of the remote transmitter device. The transitions in the incoming serial bit stream are used to re-synchronize the PPL circuitry and maintain bit and symbol lock while generating a clock recovered from the data bit stream. The serial-to-parallel converter is clocked by the recovered clock and outputs 10b symbols. The 106 symbols are clocked into the Elastic Buffer using the recovered clock associated with the receiver PLL. The Elastic Buffer is used for clock tolerance compensation; i.e. the Elastic Buffer is used to adjust for minor clock frequency variation between the recovered clock used to clock the incoming bit stream into the Elastic Buffer and the locally-generated clock associated that is used to clock data out of the Elastic Buffer.

The 10b symbols are converted back to 8b characters by the 8b/10b Decoder. The Start and End characters that frame a packet are eliminated. The 8b/10b Decoder also looks for errors in the incoming 10b symbols. For example, error detection logic can check for invalid 10b symbols or detect a missing Start or End character.

The De-Scrambler reproduces the de-scrambled packet stream from the incoming scrambled packet stream. The De-Scrambler implements the inverse of the algorithm implemented in the transmitter Scrambler.

The bytes from each Lane are un-striped to form a serial byte stream that is loaded into the receive buffer to feed to the Data Link layer.

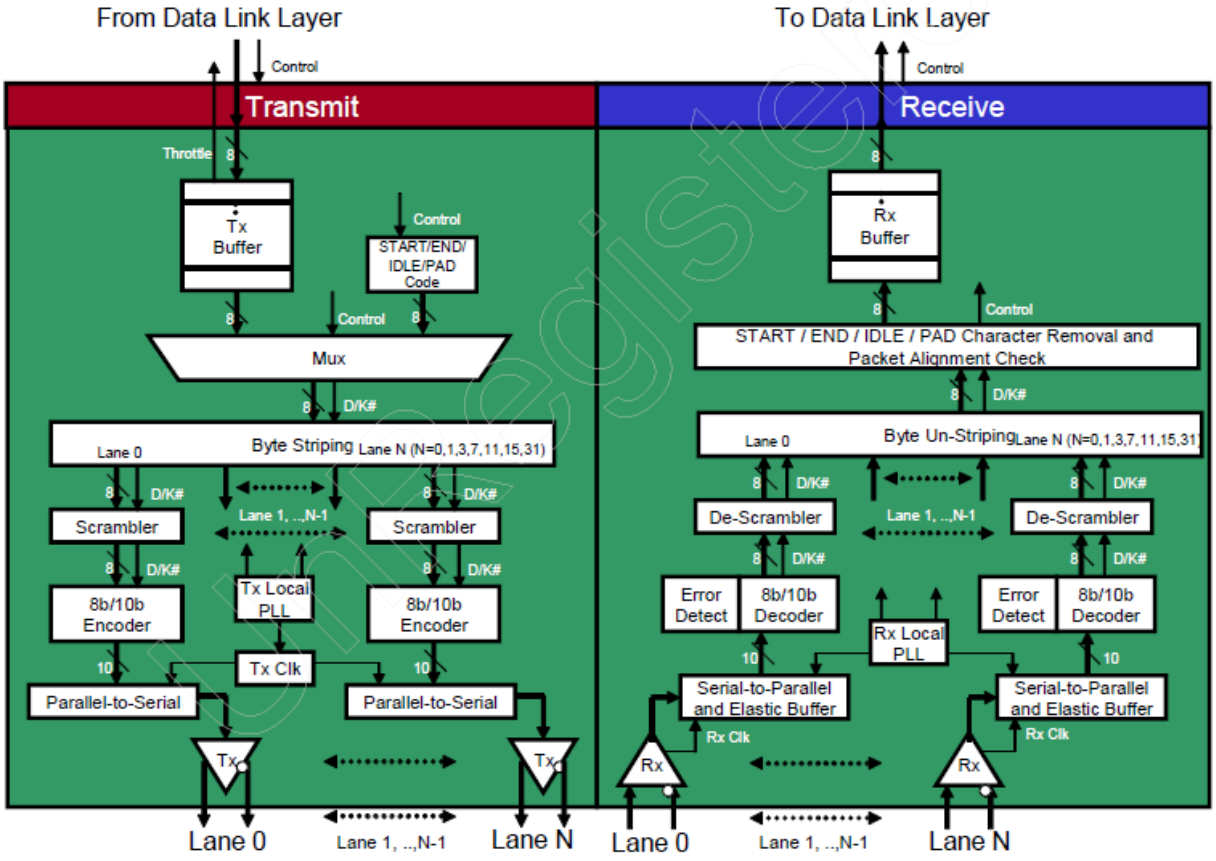


Figure 81: Detailed Physical layer

5-4 Physical layer Error Handling

When the Physical Layer logic detects an error, it sends a Receiver Error indication to the Data Link Layer. The specification lists a few of these errors, but it is far from being an exhaustive error list. It is up to the designer to determine what Physical Layer errors to detect and report.

Some of these errors include:

- 86/106 Decoder-related disparity errors.
- 8b/10b Decoder-related code violation errors.
- Elastic Buffer overflow or underflow caused by loss of symbols.
- The packet received is not consistent with the packet format rules.
- Loss of Symbol Lock (see "Symbol Boundary Sensing (Symbol Lock)" on page 441).
- Loss of Lane-to-Lane de-skew.

Response of Data Link Layer to 'Receiver Error' Indication If the Physical Layer indicates a Receiver Error to the Data link layer, the Data Link Layer discards the TIP currently being

received and frees any storage allocated for the TIP. The Data Link Layer schedules a NAK DLLP for transmission back to the transmitter of the TIP. Doing so automatically causes the transmitter device to replay TLPs from the Replay Buffer, resulting in possible auto-correction of the error. The Data Link Layer may also direct the Physical Layer to initiate Link re-training (i.e., link recovery).

Detected Link errors may also result in the Physical Layer initiating the Link retraining (recovery) process. In addition, the device that detects a Receiver Error sets the Receiver Error Status bit in the Correctable Error Status register of the PCI Express Extended Advanced Error Capabilities register set. If enabled to do so, the device sends an ERR_COR (correctable error) message to the Root Complex.

Chapter 6

Testing

6-1 Interface of layers

A Transactor, implemented in Verilog as a simple physical layer, used as a block of handshaking between two layers implemented in different languages.

As there are two layers, Transaction layer and Datalink layer, implemented in C++ and a layer, Physical layer (Transactor in this case), DPI "Direct Programming Interface" is used as the interface between them.

Figure shows how the interface between the layers looks like in the presence of DPI.

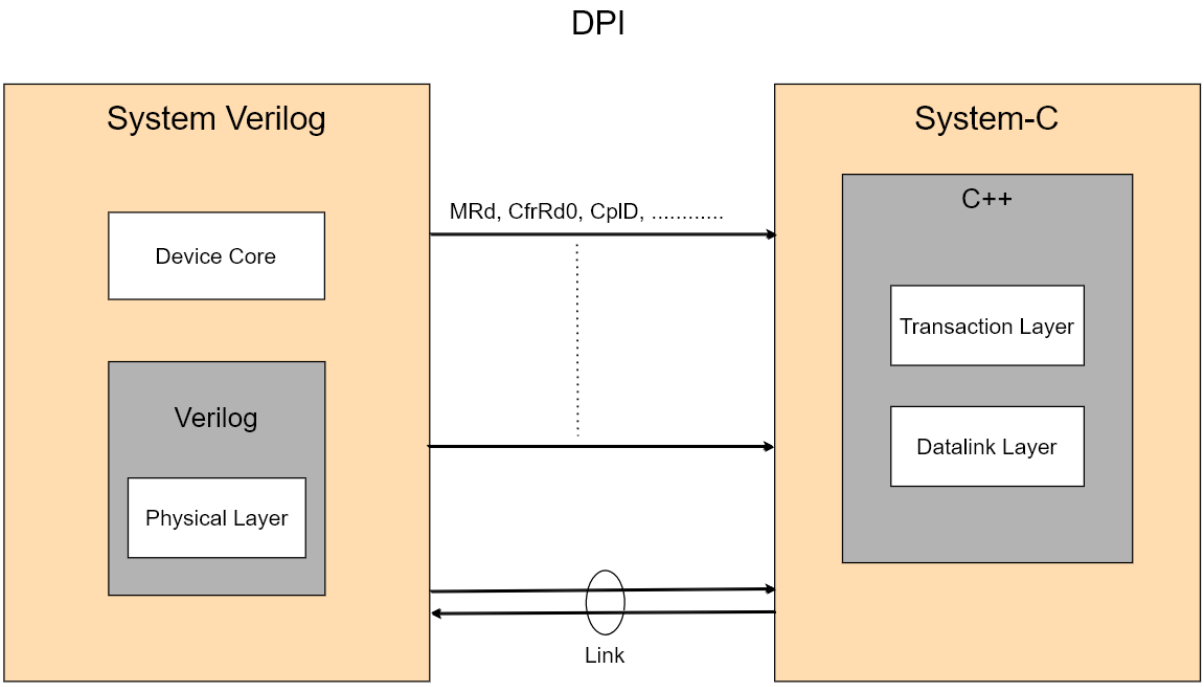


Figure 82: Layer Interface

6-2 Header type comparison

Figure show the header type 0 and **Figure** shows header type 1.

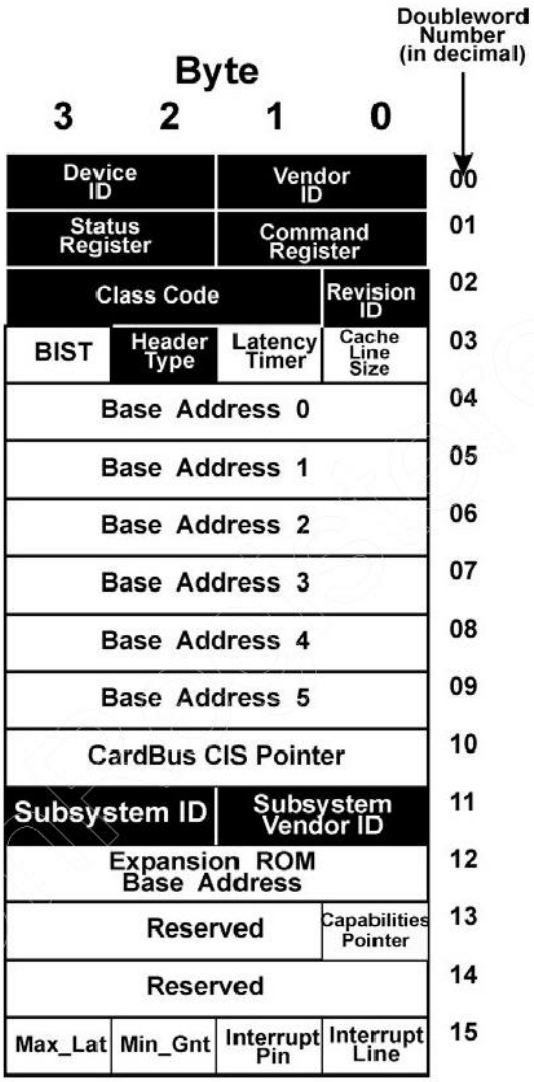


Figure 83: Header type 0

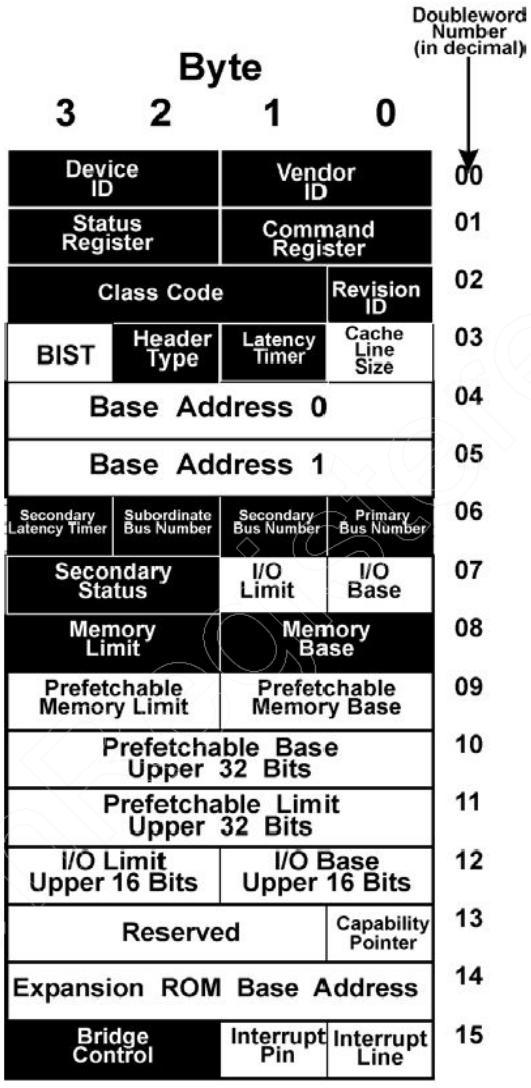


Figure 84: Header type 1

6-2-1 Header Type 0 Test

Some of the tested registers of header type 0 will be shown as follow:

- Command register.
- Status register.
- Interrupt Line register.
- Interrupt Pin register.
- PCIe Device capability
  - Device capability register.
  - Device control register.
  - Device status register.
- MSI capability
  - MSI Message Control register.
  - MSI Message Address register.
  - MSI Message Data register.
- AER capability
  - AER Enhanced header register.
  - AER Correctable status register.
  - AER un-Correctable status register.

6-2-2 Header Type 1 Test

Some of the tested registers of header type 0 will be shown as follow:

- Same registers as header type 0.
- Other registers
  - Primary Bus Number register.
  - Secondary Bus Number register.
  - Subordinate Bus Number register.
  - Memory Base register.
  - Memory Limit register.
  - Prefetchable Memory Base register.
  - Prefetchable Memory Limit register.
  - Prefetchable Upper Base register.
  - Bridge Control register.

6-3 Test types

To test the performance of the design and to know how good the implementation of the two header types, header type 0 which is used for Endpoints and header type 1 which is used for switches, that the PCIe has, two types of test is performed:

- Root Complex (OS) communicating with an Endpoint through a PCIe switch.
- Two Endpoints connected Back-to-Back.

The first type, OS communicating with an Endpoint through a PCIe switch, is used to test the design and implementation of header type 1, and the second type, two Endpoints are connected back-to-back, is used to test the implementation of header type 0.

6-3-1 First Test type

Some of the packet types tested in this method as follow:

- Configuration Write “4 Bytes”.
- Configuration Read “4 Bytes”.
- DMA (Memory Write)
  - 1 Byte.
  - 2 Bytes.
  - 4 Bytes.
  - 8 Bytes.
- DMA (Memory Read)
  - 1 Byte.
  - 4 Bytes.
  - 8 Bytes.
  - 128 Bytes.
  - 512 Bytes.

6-3-2 Packet Examples

To verify the results of the implementation, they were compared to the results of real used PCIe bus at Mentor Graphics company. Where its results were given in two separate Excel files.

Comparisons of some packets will be clarified as follow.

- Write configuration
  - Writing (0xFFFFFFFF) at BAR0.

	Expected	Output
Header 0	0x44000001	0x44000001
Header 1	0x0000CB0F	0x0000150F
Header 2	0x01000010	0x01000010
Data	0xFFFFFFFF	0xFFFFFFFF

- Completion

	Expected	Output
Header 0	0x0A000000	0x0A000000
Header 1	0x01000004	0x01000004
Header 2	0x0000CB00	0x00001500

- Read configuration
  - Reading from BAR0.

	Expected	Output
Header 0	0x04000001	0x04000001
Header 1	0x0000CC0F	0x0000160F
Header 2	0x01000010	0x01000010

- Completion

	Expected	Output
Header 0	0x4A000001	0x4A000001
Header 1	0x01000004	0x01000004
Header 2	0x0000CC00	0x00001600
Data	0xFFFFFC00	0xFFFFFC00

- Write configuration
  - Writing (0xFFFFFFFF) at BAR1.

	Expected	Output
Header 0	0x44000001	0x44000001
Header 1	0x0000CF0F	0x0000180F

Header 2	0x01000014	0x01000014
Data	0xFFFFFFFF	0xFFFFFFFF

- Completion

	Expected	**Output**
Header 0	0x0A000000	0x0A000000
Header 1	0x01000004	0x01000004
Header 2	0x0000CF00	0x00001800

- Read configuration
  - Reading from BAR0.

	Expected	**Output**
Header 0	0x04000001	0x04000001
Header 1	0x00000D0F	0x0000190F
Header 2	0x01000014	0x01000014

- Completion

	Expected	**Output**
Header 0	0x4A000001	0x4A000001
Header 1	0x01000004	0x01000004
Header 2	0x00000D00	0x00001900
Data	0xFFF00008	0xFFF00008

- Memory Test

Packet type	**#Bytes**	**Offset**	**Address**
MemRd	1 Byte	Offset 0	0xFE000000
MemRd	1 Byte	Offset 1	0xFE000001
MemRd	1 Byte	Offset 2	0xFE000006
MemWr	1 Byte	Offset 0	0xFE000000
MemWr	2 Bytes	Offset 2	0xFE000006
MemRd	4 Bytes	Offset 0	0xFE80000C
MemWr	4 Bytes	Offset 0	0xFE80000C
MemRd	8 Bytes	Offset 0	0xFE000000
MemRd	8 Bytes	Offset 0	0xFE000048
MemWr	8 Bytes	Offset 0	0xFE000000
MemWr	8 Bytes	Offset 0	0xFE100AC8
MemRd	128 Bytes	Offset 0	0x3600F800
MemRd	512 Bytes	Offset 0	0x3603FC00

6-3-3 Second Test type

Some of the packet types tested in this method as follow:

- BARs
  - 3 DWs header (32-bits address).
  - 4 DWs header (64-bits address).
- DMA.
  - Memory write.
  - Memory read.
- Multiple completions.
- Completion timeout.
- Configuration Retry Request "CRS ".
- Completion Abort "CA ".
- Unsupported Request "UR ".
- ECRC.
- MSI.

6-3-4 Packet Examples

Comparisons of some packets will be clarified as follow.

- **Multiple completions**
  - Settings
    - Command register.
      - To allow memory transactions.

Packet type	Endpoint	Data
CfgWr	E1	0x06
CfgWr	E2	0x06

    - Device status register.
      - To set Max Payload Size to 128 Bytes.

Packet type	Endpoint	Data
CfgWr	E1	0x5000
CfgWr	E2	0x5000

  - Test

Packet type	#Bytes	Direction	Address
MemRd	512 Bytes	E1 to E2	0xAC000000

    - As the MPL has been set to 128 Bytes, the completion is divided to four separate completion packets with the same sequence number.
- **Completion timeout**

- Test

Packet type	#Bytes	Direction	Address
MemRd	3 Bytes	E1 to E2	0xAC000000

➢ We had completion response delayed from App. Layer of E2 to try Completion timeout.

❖ **Configuration Retry Request "CRS ".**

➢ Sending Configuration write to E2 with wrong ID.
➢ We made the header and sent it to E1 with completion status CRS, then the correct completion has been sent to E2.

❖ **Completion Abort "CA ".**

➢ Sending Configuration write to E2 with wrong ID.
➢ We made the header and sent it to E1 with completion status CA.
➢ No need to send the successful completion as it's already terminated.

❖ **Unsupported Request.**

- Settings
  - Command register (2nd Byte).

Packet type	Endpoint	Data
CfgWr	E2	0x01

  - Device control register

Packet type	Endpoint	Data
CfgWr	E2	0x0E

➢ To enable UR & MSG error reporting.

- Test
  ➢ Sending MemWr to E2 with address 0x00F1F1F1 (wrong address) to try Msg & UR.

❖ **ECRC**

- Settings
  - AER control register

Packet type	Endpoint	Data
CfgWr	E1	0x01E0
CfgWr	E2	0x01E0

➢ To enable ECRC generation and check.

- Test.

Packet type	Endpoint	#Bytes	Address	Data
MemWr	E2	3 Bytes	0xABF1F1F1	0xFFFFFFFF
MemRd	E1	4 Bytes	0xAC000000	----------

❖ **MSI**

- Settings

Packet type	#Bytes	Register	Data
CfgWr	2 Bytes	MSI Control	0x0081
CfgWr	4 Bytes	MSI Address	0xFEE02000
CfgWr	2 Bytes	MSI Data	0x4021

➢ To enable MSI.

- Test
  - When MSI_0 is sent, INTA sends MemWr packet.

6-4 Tests in blocks

6-4-1 Transaction layer test

Figure shows the test of the Transaction layer in the first test type, OS communicating with an Endpoint through a PCIe switch, Where **Figure** shows its test in the second test type, two Endpoints are connected back-to-back.

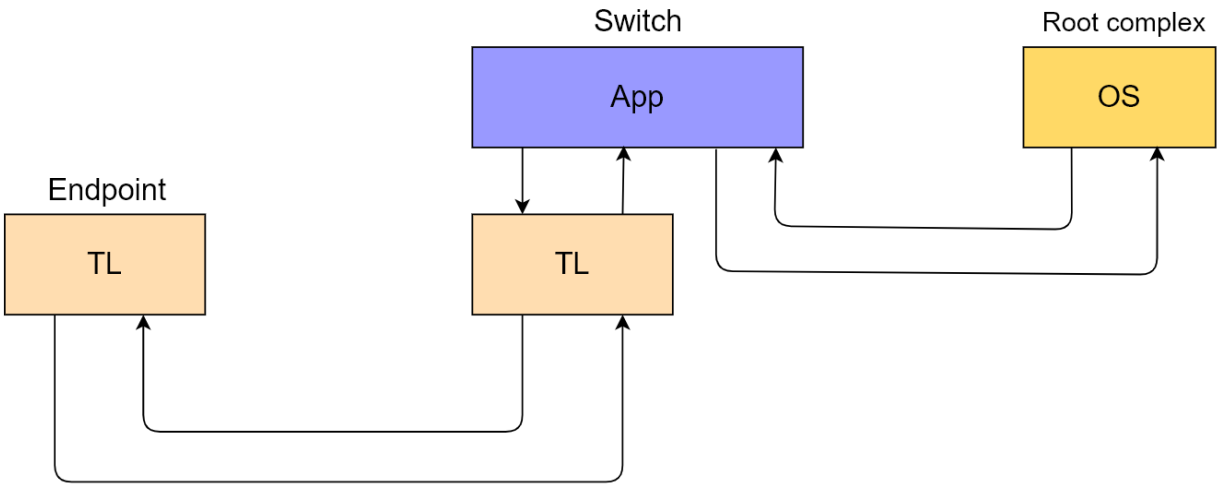


Figure 85: Transaction layer test type 1

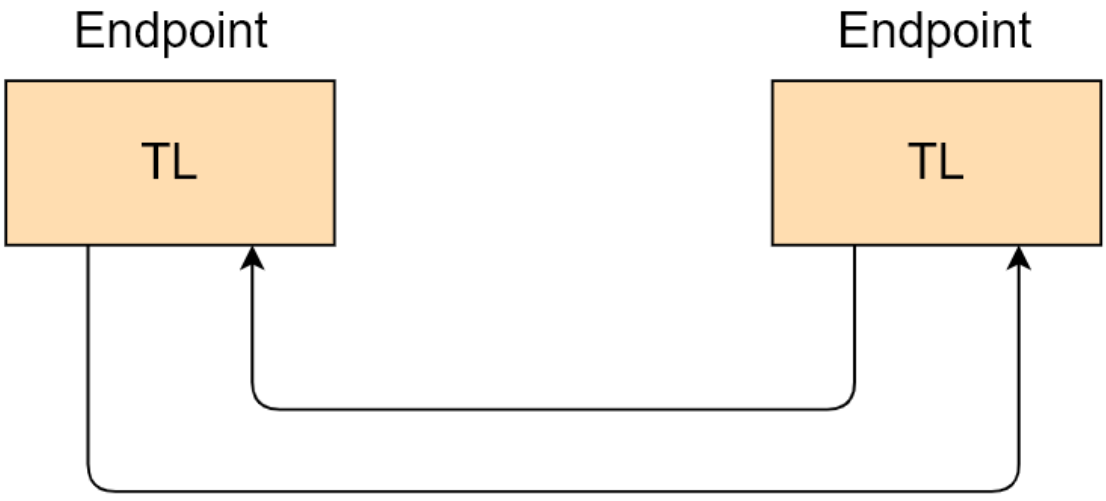


Figure 86: Transaction layer test type 2

6-4-2 Datalink layer test

Figure shows the test of the Datalink layer in the first test type, OS communicating with an Endpoint through a PCIe switch, Where **Figure** shows its test in the second test type, two Endpoints are connected back-to-back.

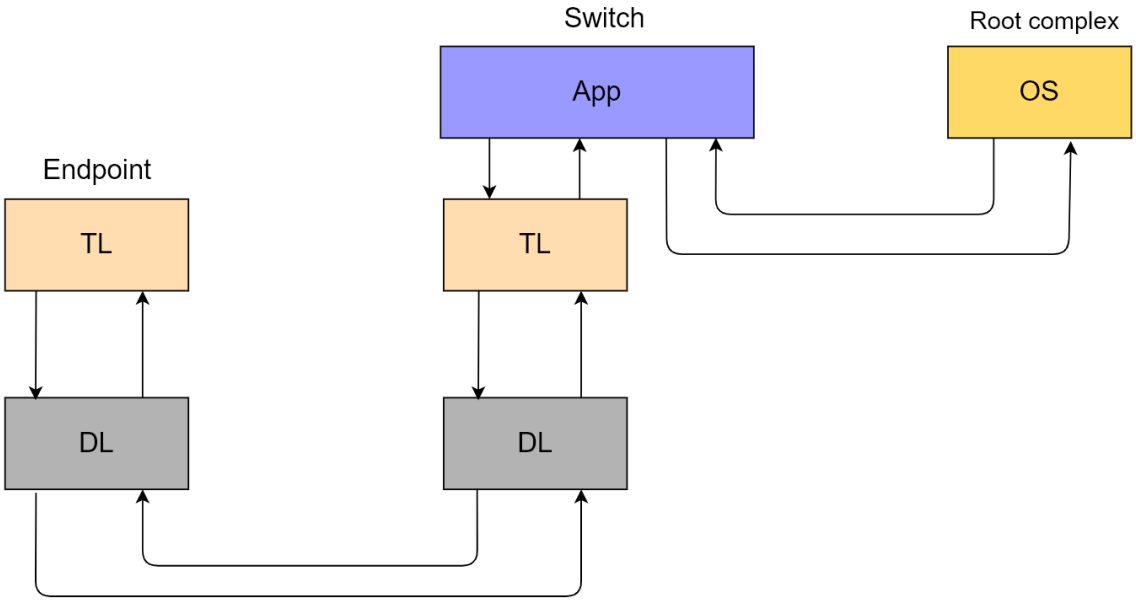


Figure 87: Datalink layer test type 1

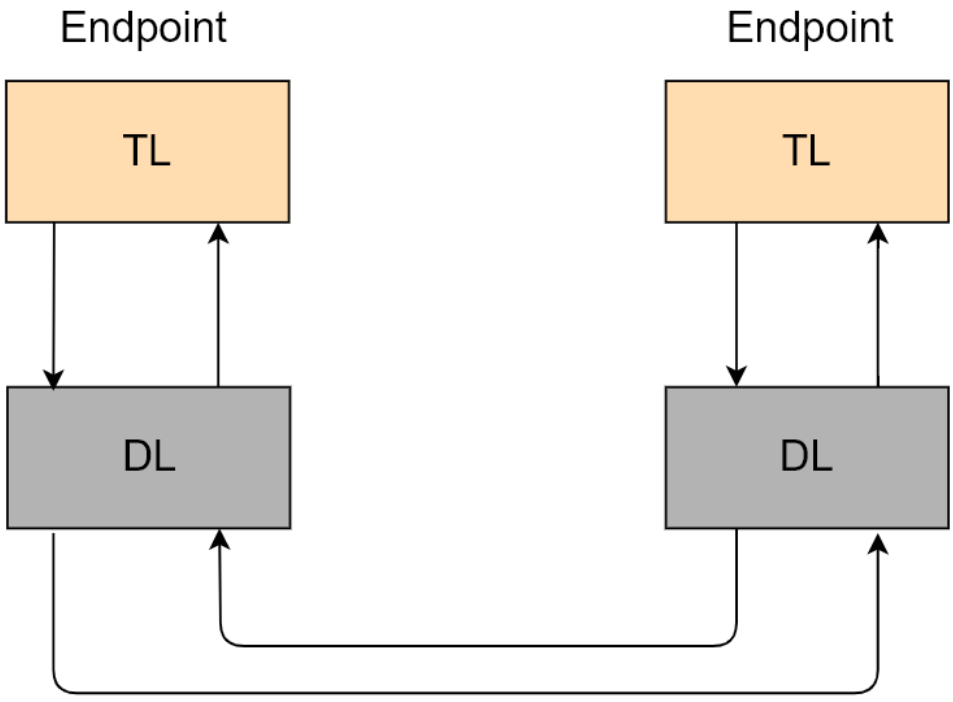


Figure 88: Datalink layer test type 2

6-4-3 Physical layer test

Figure shows the test of the Physical layer in the first test type, OS communicating with an Endpoint through a PCIe switch, Where **Figure** shows its test in the second test type, two Endpoints are connected back-to-back.

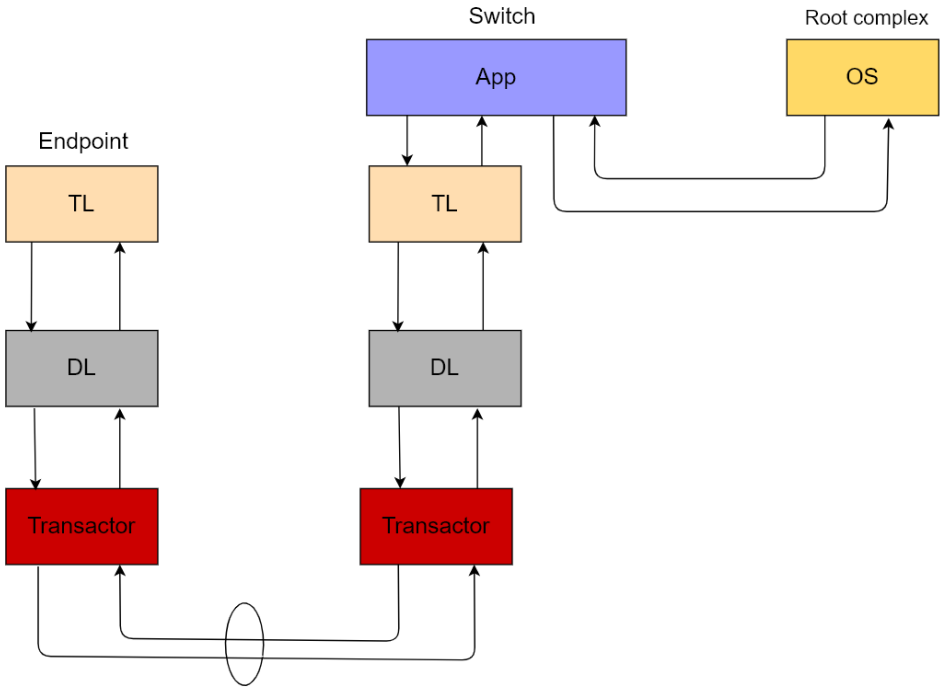


Figure 89: Physical layer test type 1

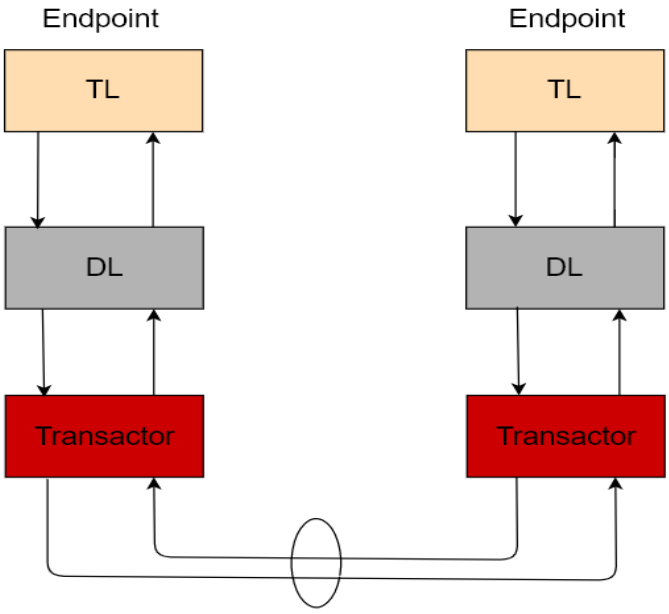


Figure 90: Physical layer test type 2

Conclusion and Future work

The main advantage of the model is it tries to reduce the complexity of the bus by the use of co-simulation. Co-simulation is simply that a part of the design is mainly hardware and the other part is mainly software, and the two parts are simulated together using some simulation tools (e.g. Questa-Sim, Model-Sim, ….). Where Questa-Sim is the tool used for this model.

So, the use of co-simulation in this model has simplified the design of the PCIe bus by Conveying some of the hardware complexity to the software part.

In the future, this model can be modified and completed in some ways. The first part to be completed is the full implementation of the Physical layer in Verilog. Then the other two layers, Transaction layer and the Datalink layer, are implemented in Verilog. Then the three completed layers of the PCIe are integrated together and tested before burning the final model of FPGA.

References

1. "PCI Express System Architecture" by MindShare, Inc, Ravi Buduck, Don Anderson, Tom Shanley.
2. https://en.wikipedia.org/wiki/PCI_Express
3. http://www.differencebetween.net/technology/difference-between-pci-and-pci-express/
4. https://en.wikipedia.org/wiki/PCI-X
5. https://community.fs.com/blog/pci-vs-pci-x-vs-pci-e-why-choose-pci-e-card.html
6. https://www.cs.unc.edu/Research/stc/FAQs/pci-overview.pdf?fbclid=IwAR2kbDOJJJdOhF7JNgBmYmDq5Z9hRebPZWCdMW3h5SwT0guxT_v7UI4QqMQ
7. https://en.wikipedia.org/wiki/Compute_Express_Link
8. https://www.mindshare.com/Learn/CXL_-_Compute_Express_Link
9. https://www.techpowerup.com/254462/intel-reveals-the-what-and-why-of-cxl-interconnect-its-answer-to-nvlink
10. Wikipedia. (2020). *Industry Standard Architecture*. [online] Available at: https://en.wikipedia.org/wiki/Industry_Standard_Architecture#History.
11. SearchWindowsServer. (n.d.). *What is ISA (Industry Standard Architecture)?* [online] Available at: https://searchwindowsserver.techtarget.com/definition/ISA-Industry-Standard-Architecture.
12. Webster, J.G. (2004). *The Measurement, Instrumentation and Sensors Handbook*. [online] Available at: http://www.kelm.ftn.uns.ac.rs/literatura/si/pdf/Measurement%20Instrumentation%20Sensors.pdf.
13. Singh, N. (2013). *All Round Experts: Difference Between 8 bit ISA, 16 bit ISA and EISA*. [online] All Round Experts. Available at: http://allroundexpert.blogspot.com/2013/12/difference-between-8-bit-isa-16-bit-isa.html.
14. Techopedia.com. (n.d.). *What is Extended Industry Standard Architecture (EISA)? - Definition from Techopedia*. [online] Available at: https://www.techopedia.com/definition/309/extended-industry-standard-architecture-eisa.
15. Akhil Ahuja (2014). Types of Buses. [online] Available at: https://www.slideshare.net/akhilahuja11/types-of-buses.
16. www.computerhope.com. (n.d.). What is VL Bus? [online] Available at: https://www.computerhope.com/jargon/v/vlbus.htm?fbclid=IwAR37139GogsWFZIXogBJyqYc7dToTDFsjW5pMSOM7i4di24bXaxxJejA8P8.
17. Wikipedia. (2020). *Industry Standard Architecture*. [online] Available at: https://en.wikipedia.org/wiki/Industry_Standard_Architecture#History.
18. Navabi, Z. and Kaeli, D.R. (2009). Computer Science and Engineering. [online] Google Books. EOLSS Publications. Available at:

https://books.google.com.eg/books?id=NLvVCwAAQBAJ&pg=PA129&lpg=PA129&dq=devices+connected+to+vesa+bus&source=bl&ots=dxVurhaiMF&sig=ACfU3U28jBnzDlUEw9oXUjyLQlis_-iI9w&hl=ar&sa=X&ved=2ahUKEwiP1uqm_JXpAhUkAWMBHSqgAtcQ6AEwDHoECAwQAQ#v=snippet&q=%20vesa&f=false.

19. morrison (2014). A history of PC buses. [online] SlideServe. Available at: https://www.slideserve.com/morrison/a-history-of-pc-buses.