CAIRO UNIVERSITY

FACULTY OF ENGINEERING
ELECTRONICS & ELECTRICAL
COMMUNICATIONS DEP.

Synthesizable RTL Physical layer for a PCIe protocol

Submitted by:

Eslam Mahmoud Ali

Esmail Hany Badr

Jala Serag El-din

Salah Abd El Khalek Mostafa

Marwa Mohamed Abd El-Aty

Mostafa Mahmoud Abozaid

Supervised by:

Dr. Hassan Mostafa

Abstract

Computer in general consists of three main components, the central processing unit CPU, memory to store data and programs, and IO devices that communicates with the outer world. These components along with the other internal components inside the computer are connected together by the means of interconnect bus. Peripheral component interconnect express PCIe, a serial high-speed bus, is the main interconnect solution used in computer nowadays. Like any protocol, PCIe provides a layered architecture of three layers, the Transaction layer, Data Link layer, and the Physical layer. Our work throughout the year was to design a synthesizable RTL implementation for PCIe logical part of the physical layer.

Keywords

Communication protocols, CPU, Processors, Buses, Parallel buses, serial buses, high speed buses, Expansion slots, Transaction layers, Data link layer, Physical layer, PIPE standard, PCI, PCI-X, AGP, PCIe, LTSSM.

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.

This project would not have been possible without the support of many people. Many thanks to our major advisor Dr. Hassan Mostafa, 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 SIEMENS EDA, Mentor Graphics, vLab PCIe team. Specially, Eng. Mahmoud El-Tahawy and Eng. Adham Rageh for providing their time and experience to help us overcome some obstacles we faced during some stages especially when dealing with SIEMENS tools and the integration with their core.

Finally, we want to thank our families, who endured this long process with us, always offering support and love. We are grateful to our colleagues and friends for always motivating us, without them we wouldn't have come so far.

Table of Contents

List of figures

List of Tables

1. Introduction

1.1. PCIe History

A computer bus is used to transfer data from one location on the motherboard to the central processing unit where all calculations take place. In our graduation project we decide to focus on one of these buses called PCIe (Peripheral component interconnect Express) bus but first let's talk about the old generation of this bus. So, first let's give an overview about PCI bus. PCI (Peripheral Component Interconnect) bus is based on ISA (Industry Standard Architecture) Bus and VL (VESA Local) Bus was developed in the early 1990's by intel, it replaced several older, slower buses that had been used early PCs. PCI quickly became the standard peripheral bus in PCs. Typical PCI cards used in PCs include: network cards, sound cards, modems, extra ports such as Universal Serial Bus (USB) or serial, TV tuner cards and hard disk drive host adapters. The PCI was a shared parallel bus topology and The PCI Bus was originally 33Mhz and then changed to 66Mhz. PCI Bus became big with the release of Windows 95 with "Plug and Play" technology. A few later, PCI-X (PCI-extended) was developed as logical extension of the PCI architecture to improve the performance but the main goal of PCI-X was maintaining compatibility with PCI devices, later the PCI-X 2.0 added new higher data rate, Table 1 shows the comparisons of frequencies of different buses topologies.

Table 1: PCI and PCI-X generations

Bus type	Clock Frequency	Peak bandwidth 32 bit-64 bit bus
PCI	33 MHz	133-266 MB/s
PCI	66 MHz	266-533 MB/s
PCI-X 1.0	66 MHz	266-533 MB/s
PCI-X 1.0	133 MHz	533-1066 MB/s
PCI-X 2.0 (DDR)	133 MHz	1066-2132 MB/s
PCI-X 2.0 (QDR)	133 MHz	2132-4262 MB/s

In 2004, a group of Intel engineers formed the Arapaho working group and started to develop a new standard. Eventually, other companies joined the group. The design went through several names before settling on PCI Express (PCIe). PCI Express (Peripheral component interconnect Express), PCIe is a high-speed serial computer expansion bus standard, designed to replace PCI, PCI-X and AGP bus standards. PCIe represents a major shift from the parallel bus model to a serial bus model, but it remains fully backward compatible with PCI in software. The initial standard, PCIe 1.0, had a data rate of 250MB/s per lane giving an aggregate rate of 2.5GT/s. The performance is usually measured in transfers per second to avoid counting overhead bits as data. PCIe 1.0 used an 8b/10b encoding scheme, The overhead bits serve two main functions. First, they ensure that there are always enough clock transitions for the serial interface to recover the clock. Second, they ensure that there is no DC current. Subsequently, there have been regular upgrades to the standard, with higher transfer rates. PCIe is used in most designs requiring a high-performance peripheral bus, no matter what the underlying architecture. PCIe 2.0, introduced in 2007, doubled the transfer rate but kept the same coding scheme. PCIe 3.0, introduced in 2010, switched to a much more efficient 128b/130b coding scheme and added scrambling for clock recovery and no DC bias. It also increased the transfer rate a lot. A 16-lane PCIe 3.0 interface could transfer 15.7 GB/s. Design for PCIe 5.0 have already started (standard was approved in May 2019) with a 32GT/s performance. Figure 1 shows the PCIe generations.

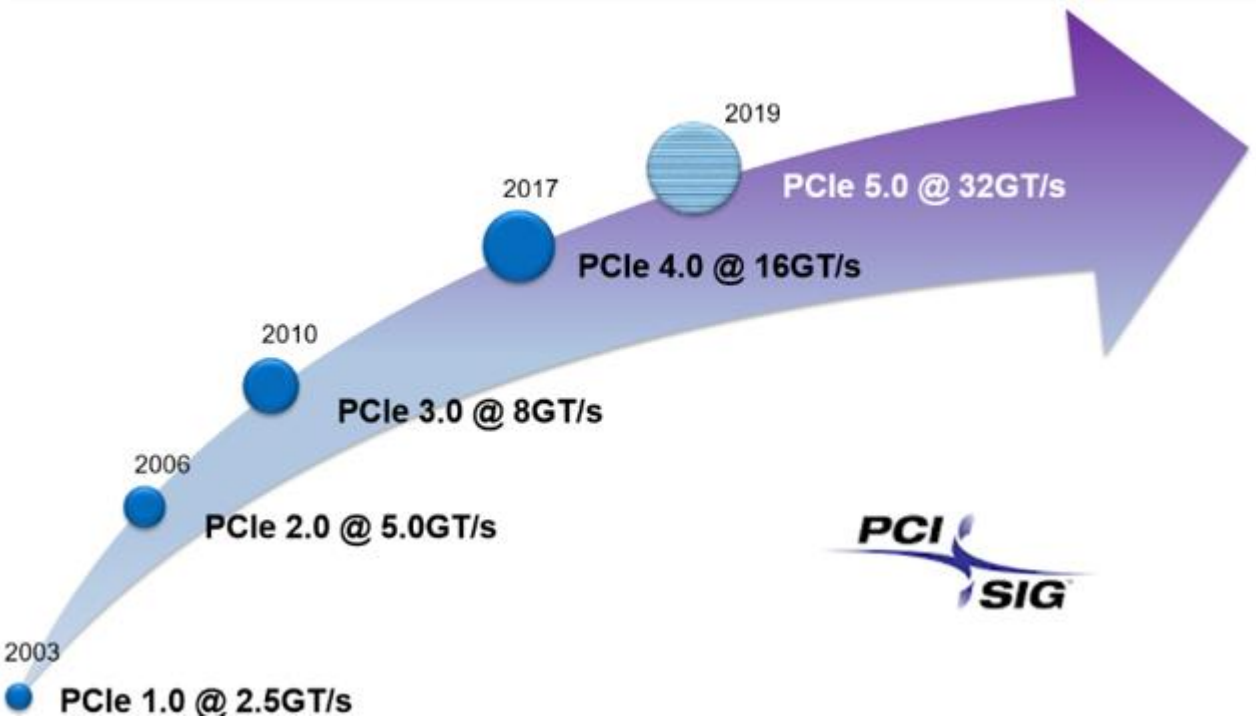

Figure 1: PCIe generations

1.2. PCIe Architecture

PCI Express is based on point-to-point topology and is capable of sending and receiving information at the same time, the path between the two devices is a **link,** Figure 2 shows PCIe devices connected through links**.** Each link is composed of one or more transmit and receive pair called a **lane,** each is a pair of differential wire, and it is capable of transmitting one byte at a time in both directions at once. The number of lanes define the link width Figure 3 shows the PCIe lane, and the PCI Express standard defines link widths of ×1, ×2, ×4, ×8, ×12, ×16 and ×32. So, more lanes will increase the bandwidth but there is a tradeoff between bandwidth, cost, and area.

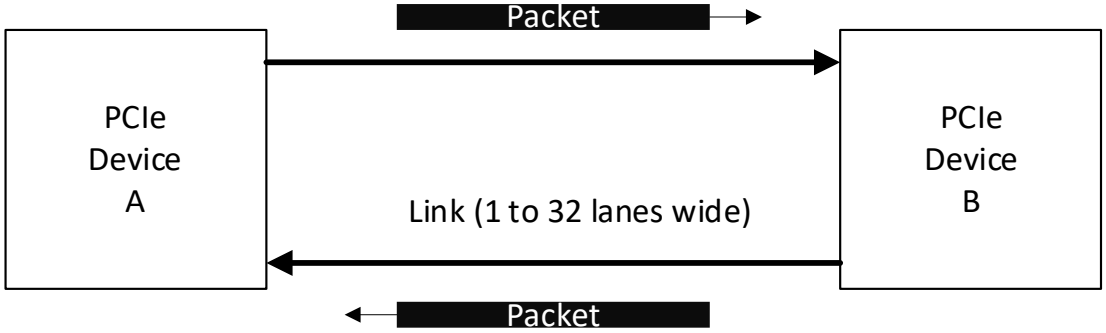

Figure 2: PCIe devices connected through Links

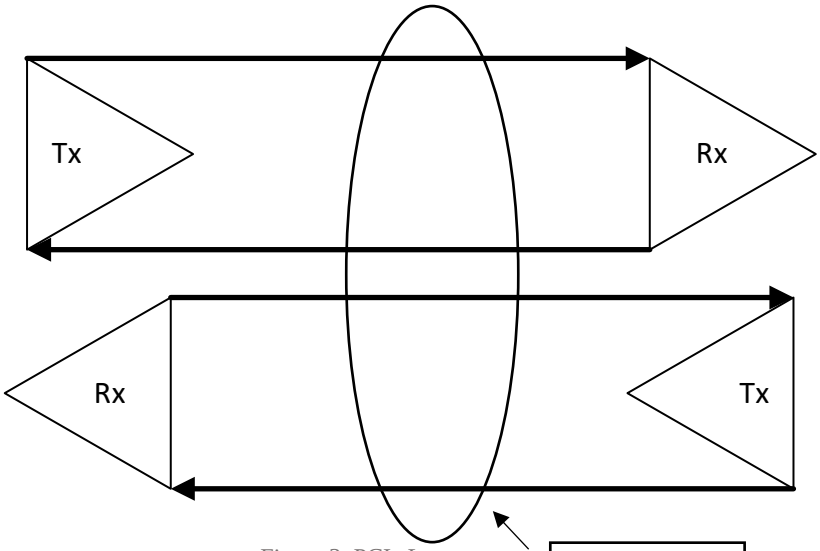

Figure 3: PCIe Lane

This allows the PCI Express bus to serve both cost-sensitive applications where high throughput is not needed, and performance-critical applications such as 3D graphics, networking and enterprise storage. The first generation of PCIe (Gen1 or PCIe spec version 1.x) is 2.5 GT/s and use 8/10 bits encoding, a process that generate a 10 bits output bases on an 8 bits input, so one lane will be able to send 0.25 GB/s in one direction, so the link total rate will be 0.5 GB/s. The second generation of PCIe (Gen2 or PCIe spec version 2.x) double the rate of Gen1 and also use 8/10 bits encoding, But Gen3 the rate increased to 8 GT/s and the encoding changed to 128/130b encoding. Table 2 show us the variations of Bandwidth with different link widths and different Generation of PCIe.

Table 2: PCIe link widths

Link width	x1	x2	x4	x8	x12	x16	x32
Gen1 bandwidth (GB/s)	0.5	1	2	4	6	8	16
Gen2 bandwidth (GB/s)	1	2	4	8	12	16	32
Gen3 bandwidth (GB/s)	2	4	8	16	24	32	64

PCIe topology and components

Figure 4 shows the PCIe topology, The CPU is considered at the top of the PCIe hierarchy, the interface between the CPU and PCIe may contains several components like processor and DRAM interfaces. **Root complex** can be defined as the interface between the CPU and the rest of the system. **Switches** allow more PCIe devices to connected to the topology. **Bridges** allow the interfaces with other buses like PCI and PCI-X to achieve the compatibility. **Endpoints** are devices in the PCIe topology not switches or bridges, they are peripheral devices such as Ethernet, USB or graphics devices. Endpoints initiate transactions as a requester or respond to transactions as a completer, there are 2 types of endpoints PCIe endpoints and Legacy endpoints to achieve the compatibility. **A Port** is the interface between a PCIe component and the Link, the port can be **Upstream** or **downstream** port. Upstream port is a port that points in the direction of the root complex so switch can be upstream or downstream but root complex always a downstream port. A Downstream Port is a port that points away from the root complex, so according to this definition an endpoint always an upstream port. **A Requester** is a device that originates a transaction in the PCIe, **A Completer** is a device addressed or targeted by a requester, The Root complex and endpoints are examples of requester and completer devices.

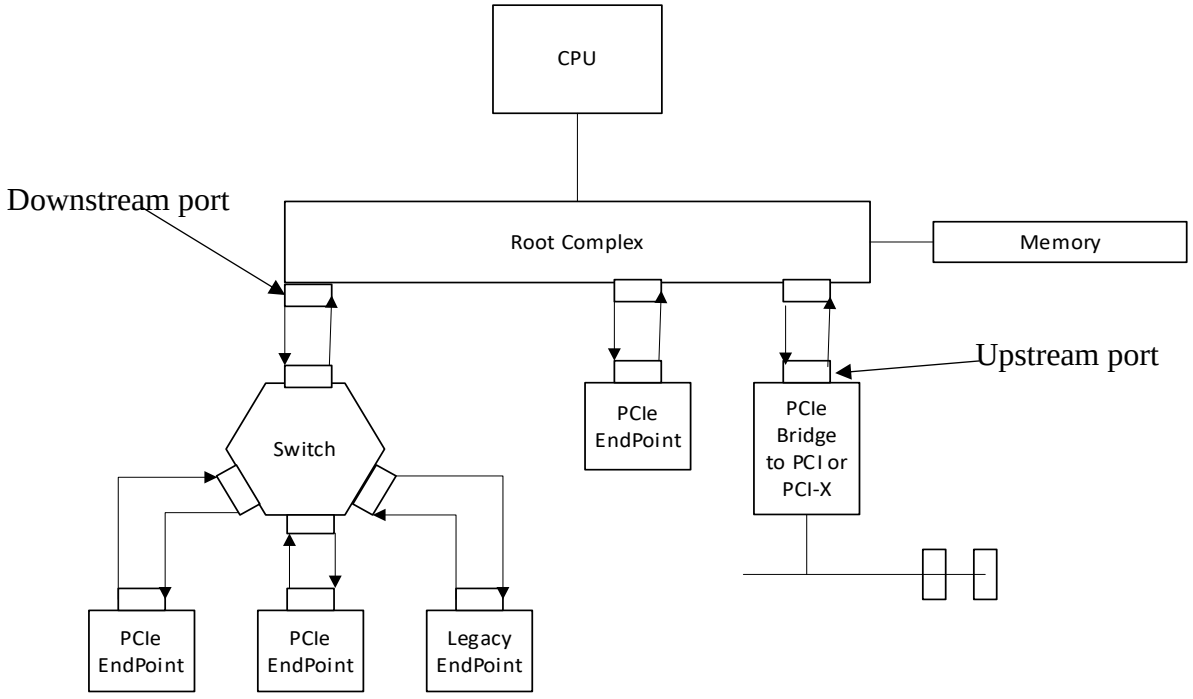

Figure 4: PCIe topology

1.3. Example for a low-cost PCIe system

Figure 5 shows us a block of low-cost PCIe system, we can note that all PCIe links are connected to the processor through the root complex. Also, we can note that the Hub link connects the root complex to ICH4 chip, which internally contains multiple switches and PCIe buses.

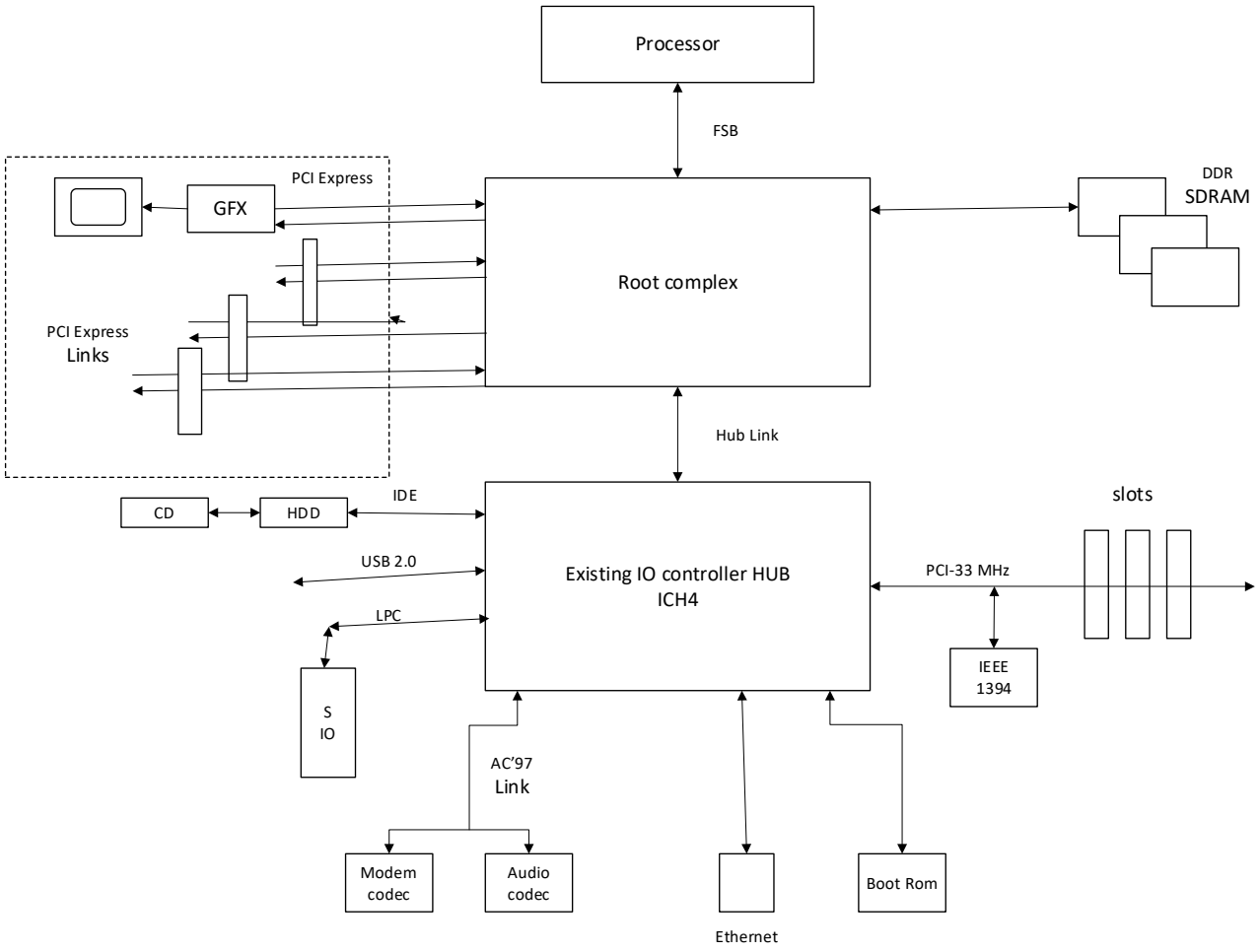

Figure 5: Example of low cost PCIe system

1.4. Configuration space

1.4.1. Bus, Devices and Functions

Before talking about configuration space, we need to discuss some PCIe basic concepts as Bus, Device and Function concept. This concept states that PCIe device consists of one or more function -up to eight functions- and each function is uniquely identified by the device it resides and the bus to which the device connects. This unique identifier is referred to as a BDF. Configuration software is responsible for detecting every Bus, Device and Function within a given topology.
A PCIe tree as shown in Figure 6 could contain up to 256 Bus, Bus 0 is typically assigned by hardware to Root Complex. Each bus could have up to 32 devices attached to it, and each device could have up to 8 functions.

1.4.2. Configuration address space

PCIe inherited the concept of implementing standardized configuration registers that permit operating systems to manage virtually all system resources.
Each function has its 4KB configuration address space, this is divided into a 256 bytes PCI-compatible space, that is compliant with legacy PCI software, the first 64 bytes of it is called the

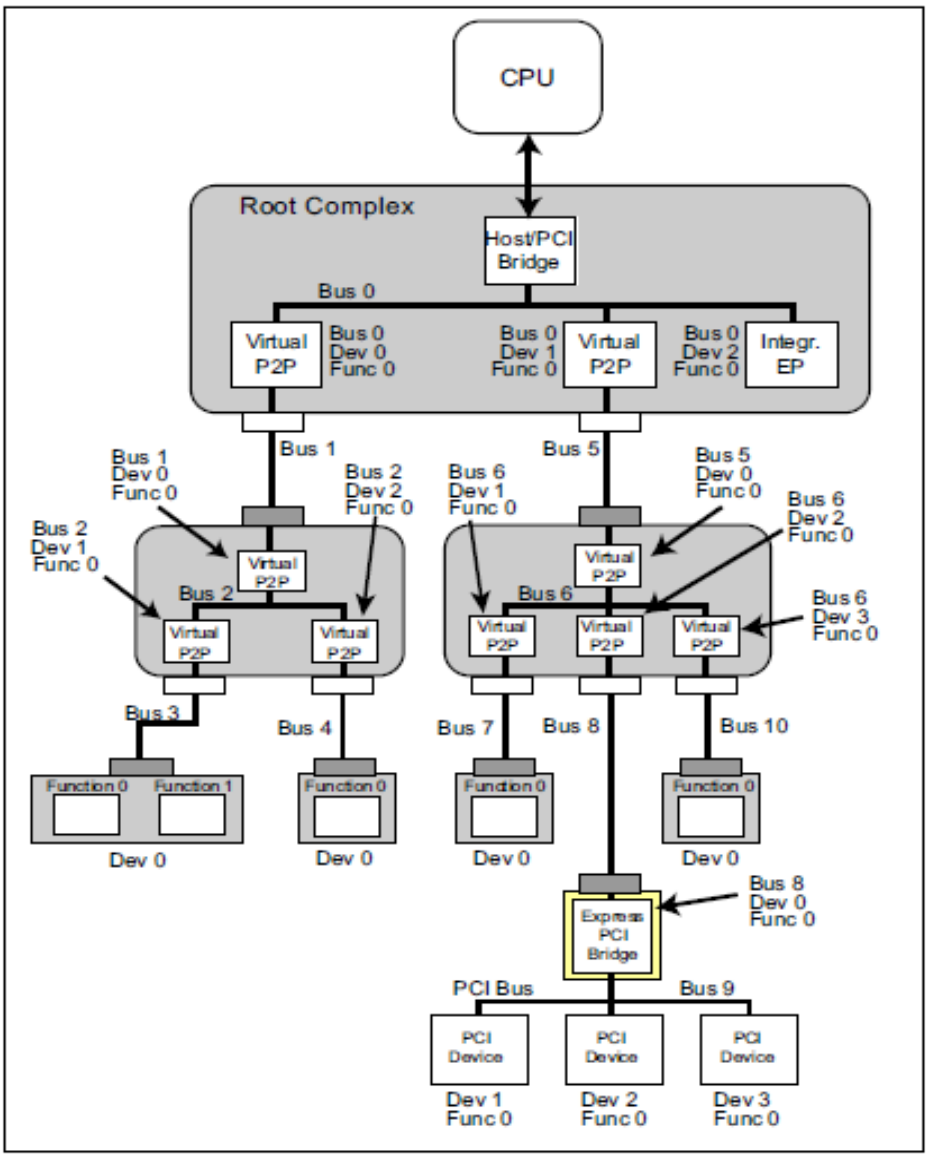

Figure 6: PCIe tree

configuration header. Two types of headers exist, Type 0 for every function that is not a bridge function which implements Type 1 header. The rest of the 4KB goes to Extended configuration space that implements more features needed in PCIe. Figure 7 shows the description for the configuration space header types and contents.

Configuration space is accessed through configuration requests, which are the non-posted configuration reads and configuration writes TLPs mentioned in the Device layers section. Only the Root Complex could initiate these types of TLPs, and the completer, which in this case is an Endpoint or a switch, should respond with the proper completion.

These configuration requests and completions are routed using ID routing, which is the routing based on the BDF concept.

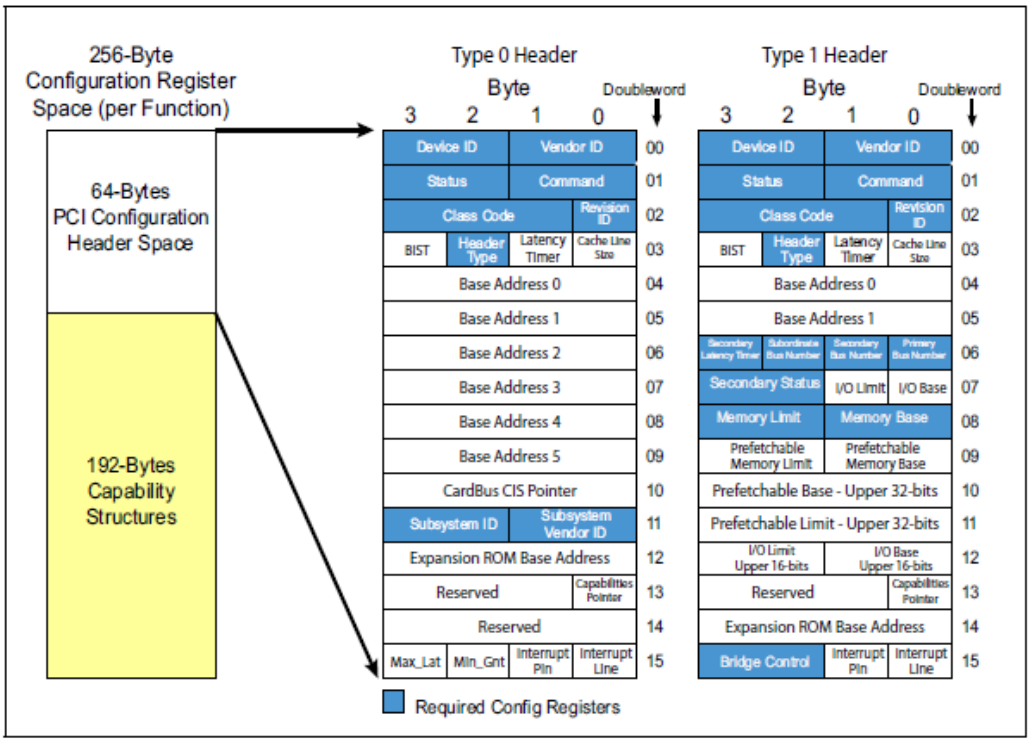

Figure 7: Description of configuration space header

1.5. Enumeration process

After system reset or power up, configuration software has to scan the PCIe fabric to discover the existing devices, defines that it is an endpoint or a bridge, and give each function its BDF number. This process is called the enumeration process.

1.6. CXL Bus

1.6.1. CXL overview

Compute Express Link is a cache-coherent interconnect for processors, memory expansion, and accelerators that maintains a unified coherent memory space between the CPU and any memory on the attached CXL device. The Compute Express Link (CXL) is an open industry-standard interconnect offering coherency and memory semantics using high-bandwidth and low-latency connectivity between host processor and devices such as accelerators, memory buffers, and smart I/O devices. CXL is based on the PCI Express (PCIe) 5.0 physical-layer infrastructure with plug-and-play interoperability between PCIe and CXL devices on a PCIe slot. The next figure show that the CXL physical layer is a PCIe based.

1.6.2. Difference between CXL and PCIe

Figure 8 shows us the difference between PCIe layers and CXL layers.

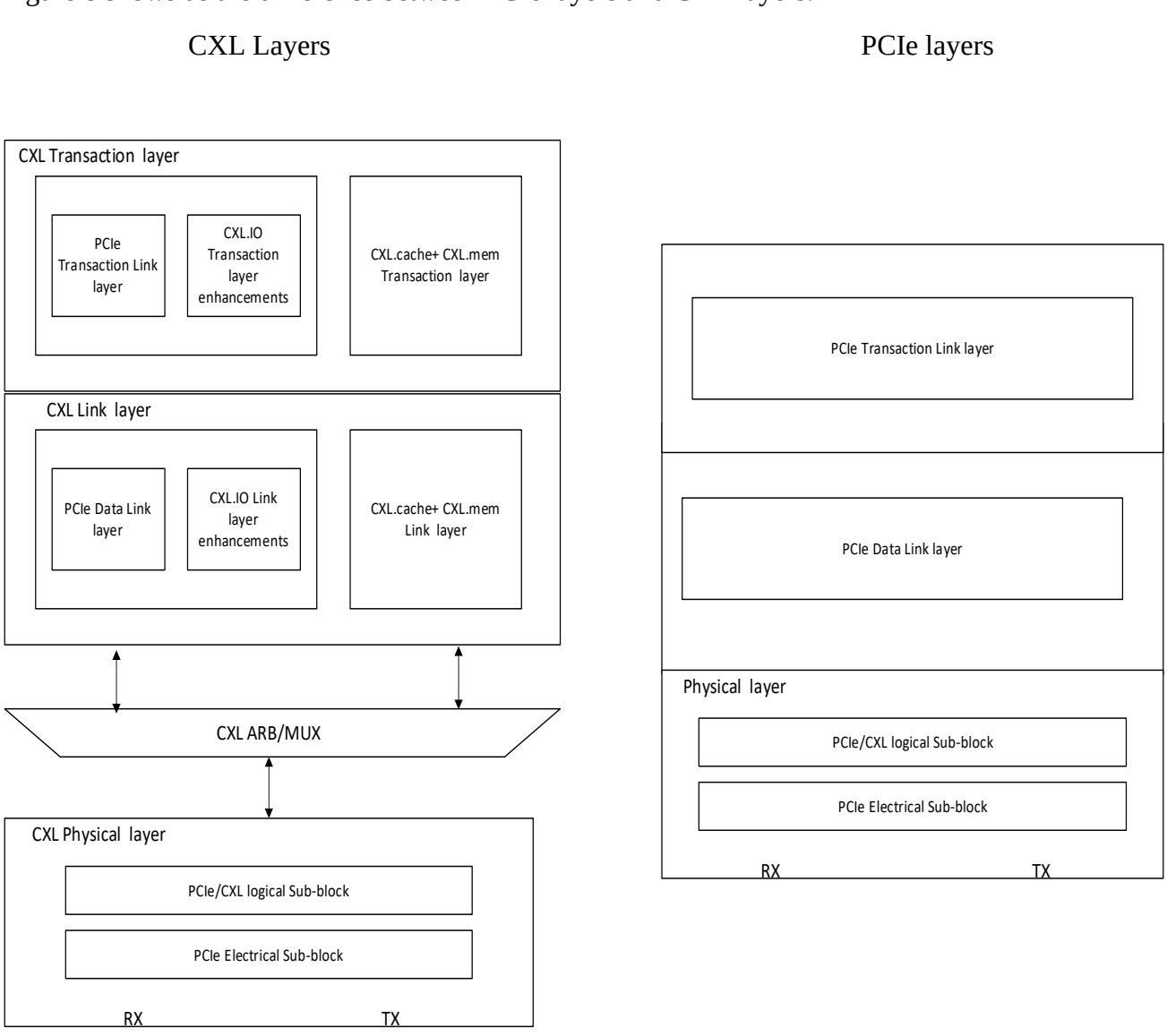

Figure 8: Difference between CXL layer and PCIe layers

Table 3 shows us some point of comparisons between PCIe and CXL.

Table 3: PCIe and CXL comparisons

Point of comparisons	PCIe	CXL
Bandwidth	Smaller than CXL	High Bandwidth compared to PCIe
Protocol's support	IO protocol	• CXL.mem • CXL.Cache • CXl.IO
Large number of devices	PCIe fails (Perfect for client segment device).	CXL Prevails (Perfect for data center)
Design complexity	PCIe is simpler	CXL is more complex

1.7. Device layers

The PCIe architecture is defined as a layered architecture as shown in Figure 9. Each layer can be spilt into two logical parts. The first one is the transmitter part which sends the outbound packets while the second one is the receiver part that receive the inbound packets. Above the main three layers, there is another layer called the device core or the software layer. This implements the core of the device that determine its functionality. If the device is an end point, it may consist of up to 8 functions. Each function has its own configuration space. If the device is a switch, it contains the packet routing logic. If the device is a root complex, root core implements a virtual PCI bus 0.

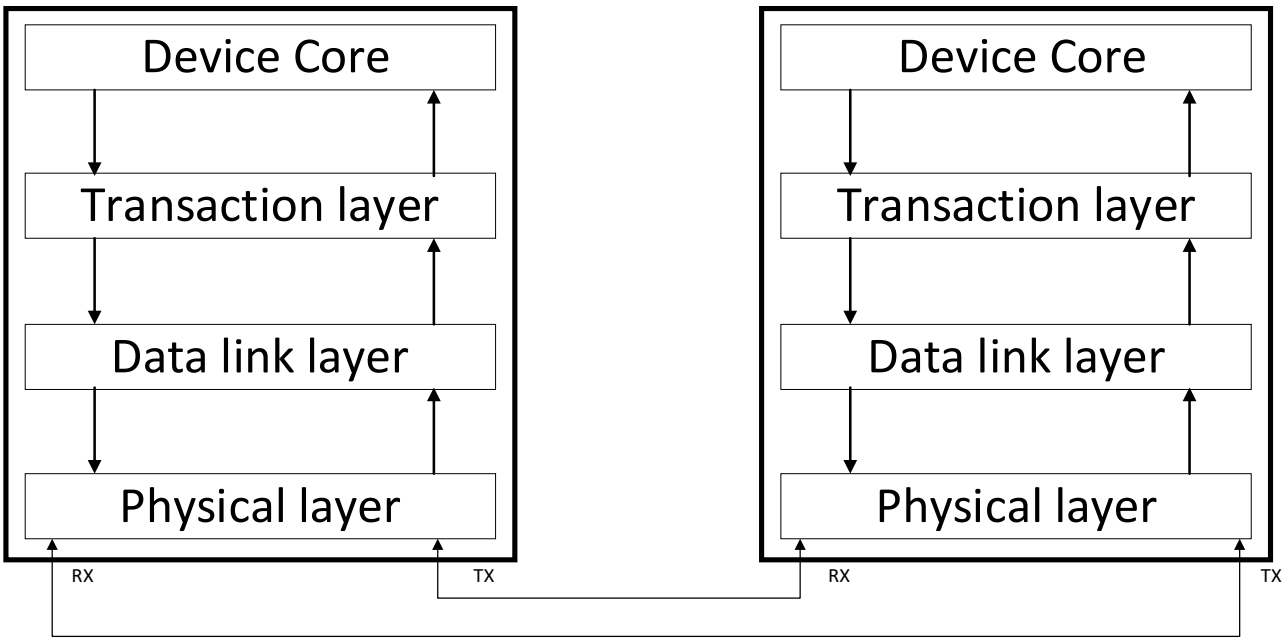

Figure 9: PCIe layers

1.7.1. Transaction layer

The transaction layer is responsible for TLP (Transaction layer packets) assembly and disassembly creation and decoding. The TLPs handled the transaction, and the transaction is defined as a combination of request. The requests can be posted or non-posted. The posted request means that the device targeted does not return a completion to the requester, but the non-posted request means that the device targeted return a completion to the requester. The transaction layer is also responsible for flow control functionality and transaction ordering functionality. The flow control functionality is that each device sends the amount of free space in its received buffer to the other device. This prevents the sender device from sending a packet with size larger than the free space in the received device.

1.7.1.1. TLP assembly and disassembly

There are 4 categories of requests, The first three already handled by PCI and PCI-X but messages are new type for PCIe.

1. Memory requests: memory transactions include 2 classes, read requests with their completions and write requests. these types of transactions are routed through memory addresses.
2. IO requests: IO transactions used for legacy devices.
3. Configuration requests: configuration requests used to access the configuration space to access the device, include 2 types, type 0 everything in the topologies of PCIe, except the switches and the bridges.
4. Messages requests: Message request is a new type defined in PCIe, include Power management, Error signaling etc….
5. Completions: completions are expected in response for non-posted requests

Each request must include

1. Target address or ID
2. Requester ID
3. Transaction type, for example memory write
4. Data payload size
5. Traffic class (used for packet priority)

TLP assembly

The device core sends the data required to assemble it in the TLP, each TLP will have a header to indicate some information like the requester address and the target address, also each TLP will have an optional ECRC to check the bits error. Figure 10 shows the sent TLP packet.

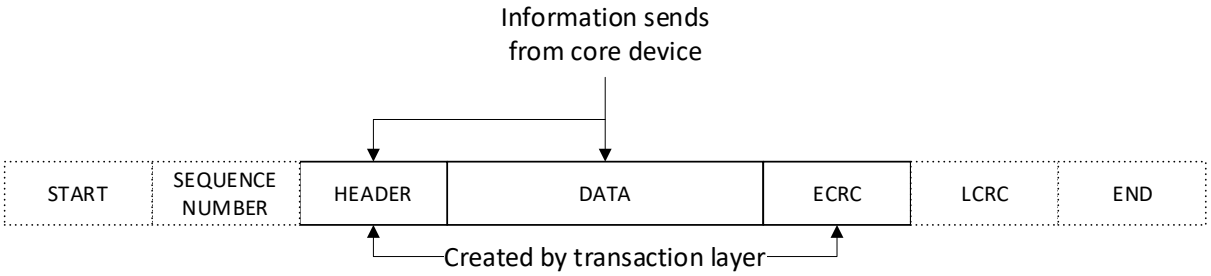

Figure 10: sent TLP packet

TLP disassembly

When the receiver receives the TLP in transactions layer, the TLP is decoded and the information is passed to the core logic. As shown in Figure 11.

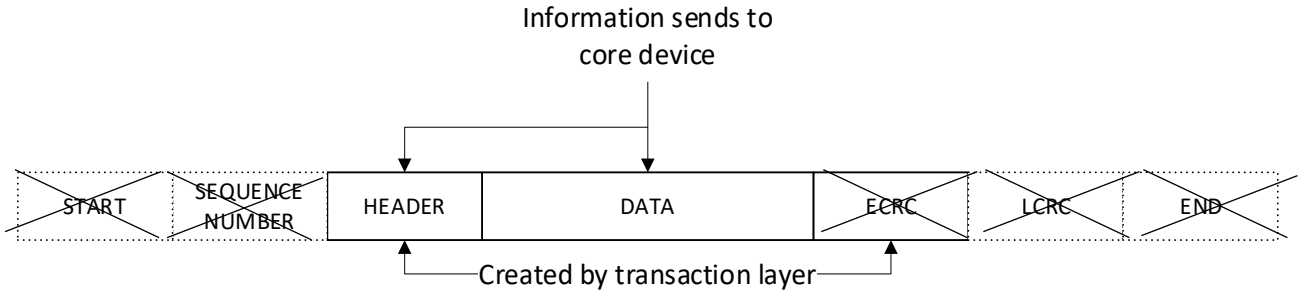

Figure 11: received TLP packet

TLP components

Table 4: TLP components

TLP component	Protocol layer	Component use
Header	Transaction layer	3 or 4 double word (12 or 16 bytes), the header defines the parameters: • Transaction type • Target address and ID • Transfer size • Attributes • Traffic class
Data	Transaction layer	Optional 1-1024 double word payload
ECRC	Transaction layer	Optional 1 double word

1.7.1.2. Flow control

The flow control is a mechanism uses a credit-based mechanism that allow the transmitting port to be aware of buffer space available at the receiving port, to sends the TLPs with no losses. The credits are updates using the flow control DLLPs, also we need to notes that the flow control is a shared responsibility between the data link layer and the transaction layer, The data link layer sends and the receive the information about the buffer space but the transaction layer contains the counter that counts the available space, so we can summaries the process of the flow control into 3 parts

- Devices report available buffer space
- Receivers register credits
- Transmitters check credits

1.7.2. Data link layer

On the transmit side, Data link layer receives the TLPs from the transaction layer, adds a sequence number to it and calculate a CRC and append it to the upcoming TLP, then passes it to the physical layer. On the receive size, it removes the sequence number and check the CRC for errors, then passes it to the transaction layer to decompose it.

Data link layer also form its own packets (DLLPs). DLLPs have a simple packet format and are of a fixed size, 8 bytes total (Figure 12) including the framing bytes added in the physical layer. They are local traffic which means that they are not routed but communicated between the nearest-neighbor device.

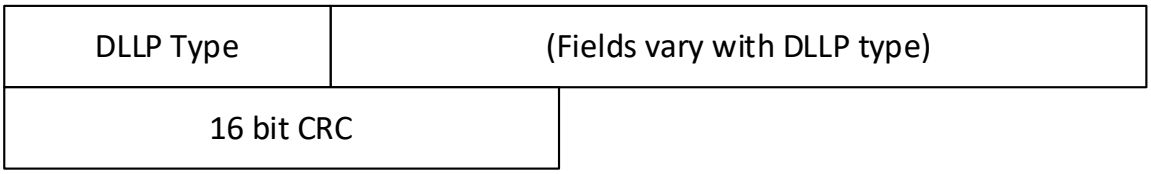

Figure 12: DLLP format

DLLPs are used for Ack/Nak protocol, power management, and flow control. Ack/Nak protocol is used to insure the TLPs successful transmission. When a TLP is sent, the data link layer adds a sequence number to it and saves a copy of this TLP in a buffer called the reply buffer. To ensure that this TLP is delivered successfully to its intended destination, data link layer of the sender awaits an Ack or a Nak from the target with the same sequence number sent with the TLP. If an Ack is received, the TLP copy in the reply buffer is discarded and the TLP is considered successfully reached its destination, if a Nak is received, retransmission of the TLP take place. DLLPs are also used to negotiate the flow control credits of the TLPs as mentioned in the Transaction layer section.

1.7.3. Physical layer

Physical layer of PCIe consists of two parts, the logical part, and the electrical part. The interface between the two parts occurs through the PIPE standard. The electrical part is responsible for signaling, the electrical levels, data integrity and other things which are not our concern in this context. The logical part, is responsible for link training and initialization, receiving the data from the data link layer, encodes it using 8b/10b encoding (Gen1), add framing character, called start and end characters, according to the type of the received data from Data link layer, whether it is a TLP or a DLLP, as shown in Figure 13. The physical layer originates its own packets but they are called Ordered-Sets, these ordered sets has several types as the Training sequences TS1 and TS2 which are used in link training process. The Electrical idle and Electrical idle exit ordered sets, which are used to enter and leaving the low power states to save power. The skip SKP ordered sets, which are used in the receiver clock compensation.

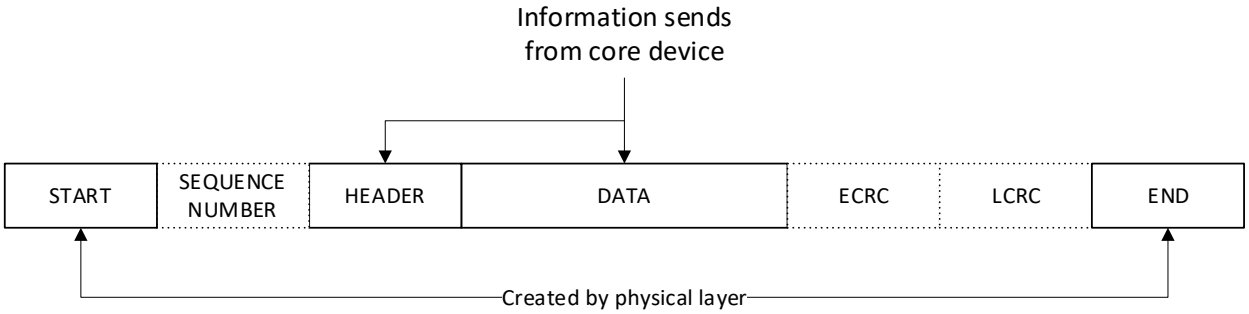

Figure 13: framing characters added by physical layer

Our work is to implement the logical part of the physical layer as a synthesizable RTL. This layer is divided into three parts, the Tx block, Rx block and LTSSM controller block as shown in Figure 14.

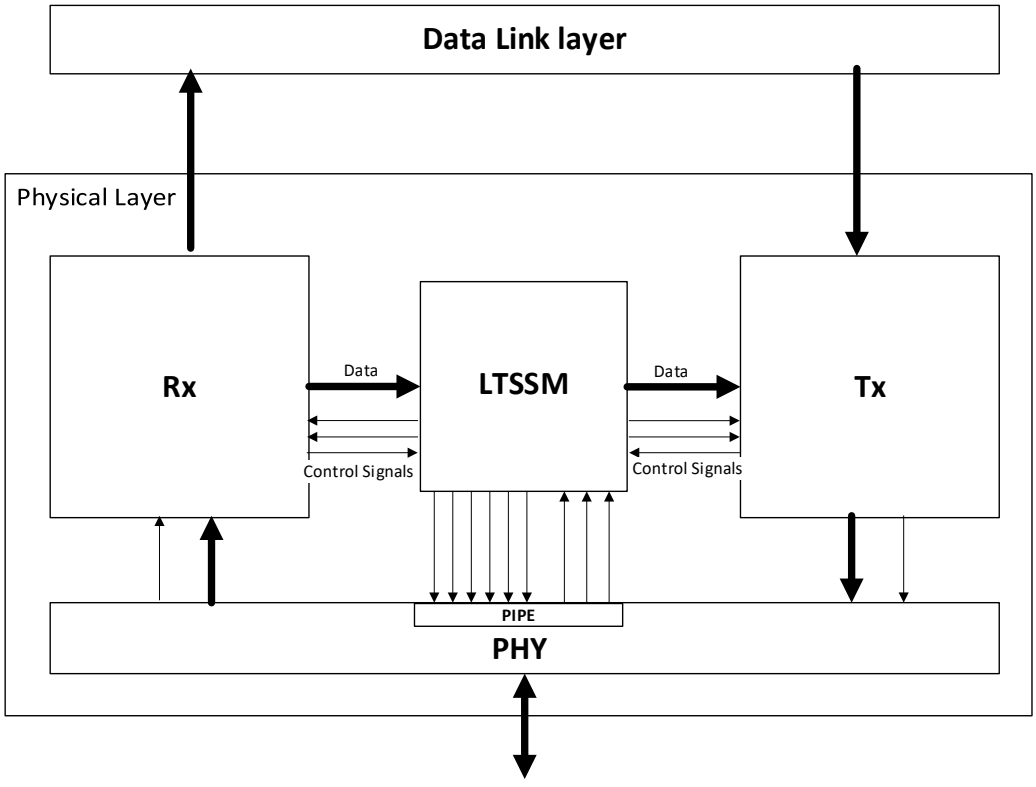

Figure 14: physical layer logical part implementation

2. Implementation

2.1. LTSSM

2.1.1. LTSSM overview

Link training and status state machine (LTSSM) is the hardware block which is responsible of the full training and initialization process of the PCIe link, this process should take place to insure normal packet traffic on the link. This process is automatically initiated by hardware after device reset.

For Gen1, two main things are configured during link initialization and training process which are:

Bit Lock: at the beginning of training process, the receiver's clock is not yet synchronized with the transmit clock of the incoming signal. Using the incoming bits stream, clock and data recovery (CDR) of the receiver recreates the transmit clock. Once this operation is done, Bit Lock is said to be acquired.

Symbol Lock: after acquiring Bit Lock, the receiver can detect the incoming bits correctly but not the beginning of each symbol. Symbol Lock can be achieved searching for a unique pattern called COM symbol, which is send at the beginning of each training sequence ordered set (TS1s or TS2s). once the COM symbol is recognized, Symbol Lock is said to be acquired.

There are other configurations that happen during link training and initialization process, but they are not important in the Gen1 x1 implementation.

Ordered-Sets

Link training and initialization process could not take place without the presence of the Training sequence Ordered-Sets (TS1s and TS2s). the Gen1 TS1s and TS2s consist of 16 symbols as shown in Figure 15, they are exchanged in the Polling, Configuration and Recovery states (LTSSM states).

Details of TS symbols is as follows:

Symbol 0: COM character, the first symbol of any ordered set. Receivers use this character to acquire symbol lock. It could be used in lane de-skewing as it appears on all lanes.

Symbol 1: Indicates the "Link", takes the value of PAD character in Polling state, and set to the link number in other states.

Symbol 2: Indicates the "Lane", takes the value of PAD character in Polling state, and set to the lane number in other states.

Symbol 3: Indicates the number of fast training sequences "N_FTS" required by the receiver to exit L0s state and reach L0 state.

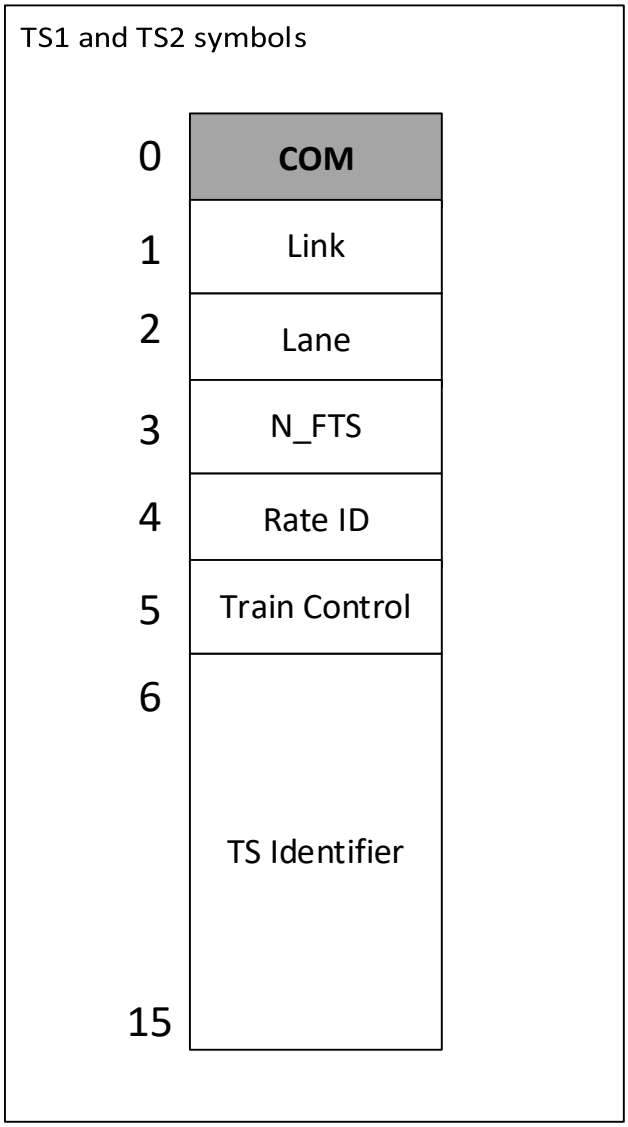

Figure 15: TS contents

Symbol 4: Indicates the "Rate ID", devices report the data rates supported by them, along with a little more information used for hardware-initiated bandwidth changes. The Gen1 speed "2.5 GT/s" must be always supported and link will always train to that speed to insure backward compatibility of new devices with older ones.

Symbol 5: Indicated the "Train Control" symbol, which communicate special conditions such as Hot Reset, Enable Loopback mode, Disable Link, and Disable Scrambling. These conditions are not supported in the implementation discussed in this book, and Symbol is set to all 0's.

Symbol 6-15: Indicates the TS1 or TS2 identifiers, for TS1 it takes the value of 4Ah and for TS2 it takes the value of 45h.

2.1.2. LTSSM interface with Transmitter, Receiver block and Pipe

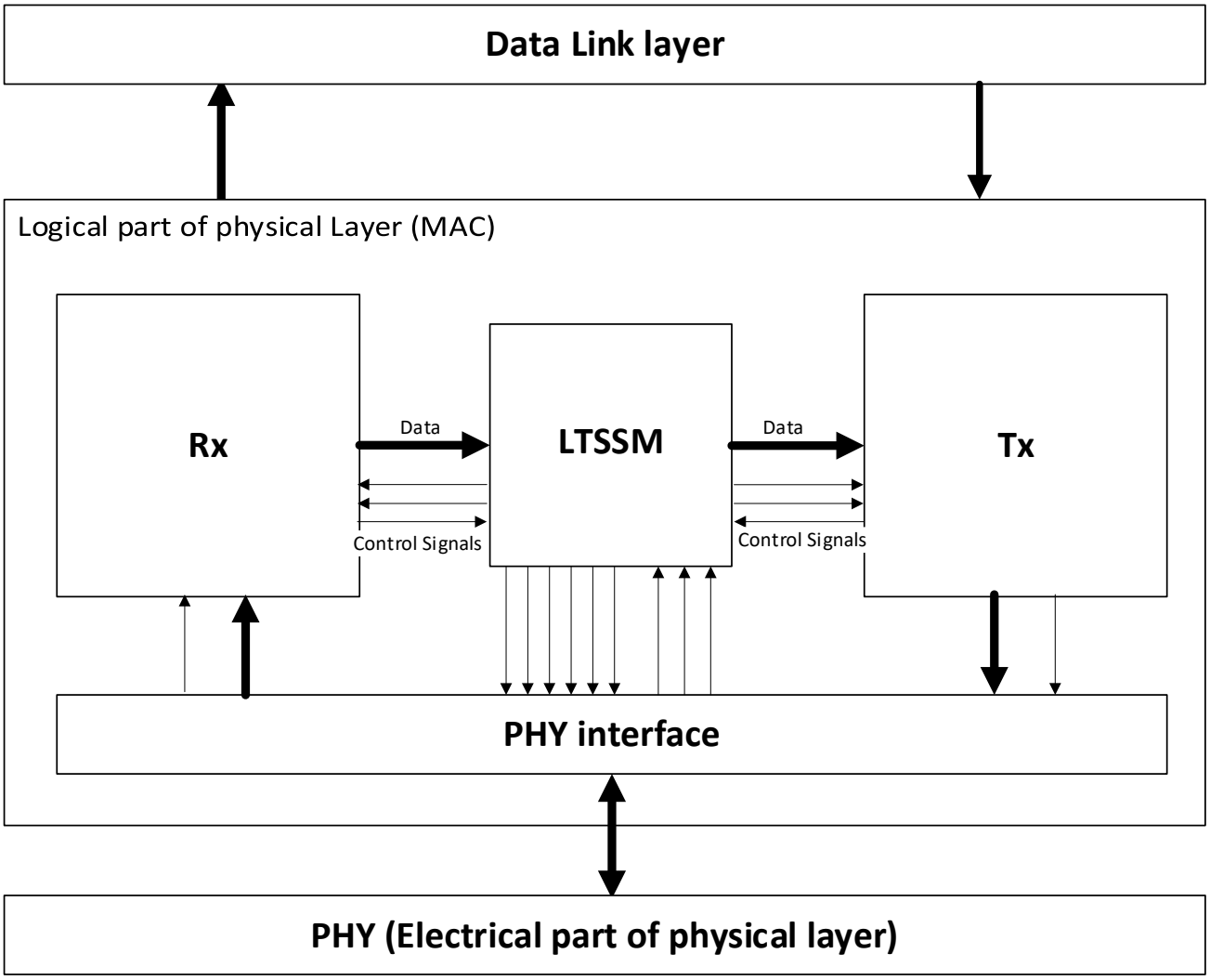

Figure 16: LTSSM interfaces

The LTSSM has three interfaces as shown in Figure 16, one with the PCIe Tx, another with PCIe Rx and the last with electrical part of physical layer (**PHY**). The interface with the PHY is done according to PHY Interface for the PCI Express (**PIPE**) standard. In this subsection, a detailed description of these interfaces is presented.

2.1.2.1. PHY interface

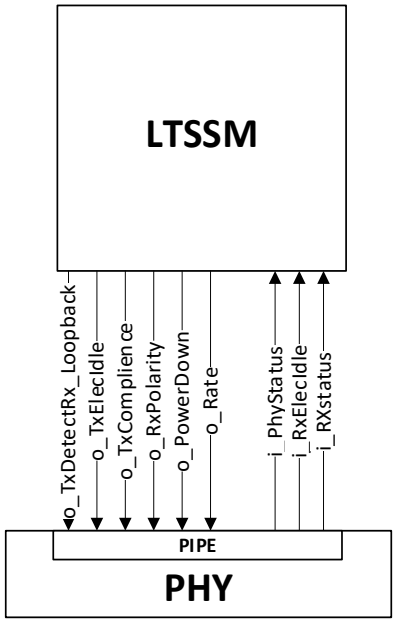

Figure 17: LTSSM-PHY interface

Table 5: LTSSM-PHY interface signal

Signal	Direction	Description
i_PhyStatus	Input	**Inputs from PHY** Used to communicate completion of several PHY functions including stable PCLK after Reset_n deassertion, power management state transitions, rate change, and receiver detection.
i_RxElecIdle	Input	**Inputs from PHY** Indicates receiver detection of an electrical idle.
i_RxStatus[2:0]	Input	**Inputs from PHY** Encodes receiver status and error codes for the received data stream when receiving data. • RxStatus[2:0]=000, Received data OK. • RxStatus[2:0]=001, 1 SKP added. • RxStatus[2:0]=010, A SKP removed. • RxStatus[2:0]=011, Receiver detected. • RxStatus[2:0]=100, Both 8B/10B decode error and receive disparity error. • RxStatus[2:0]=101, Elastic buffer overflow. • RxStatus[2:0]=110, Elastic buffer underflow. • RxStatus[2:0]=111, Receive disparity error.
o_TxDetectRx_loopback	Output	**Output to PHY** Used to tell the PHY to begin a receiver detection operation or to begin loopback
o_TxElecIdle	Output	**Output to PHY** Tells PHY that the transmitter is electrically idle

o_TxCompliance	Output	**Output to PHY** Sets the running disparity to negative. Used when transmitting the PCI Express compliance pattern.
o_RxPolarity	Output	**Output to PHY** Tells PHY to do a polarity inversion on the received data.
o_PowerDown[1:0]	Output	**Output to PHY** Power up or down the transceiver • PowerDown[1:0]=00, L0, normal operation. • PowerDown[1:0]=01, L0s, low recovery power saving state. • PowerDown[1:0]=10, L1, long recovery power saving state • PowerDown[1:0]=11, L2, lowest power state.
o_Rate	Output	**Output to PHY** Control the link signaling rate.

The LTSSM interfaces with the PHY layer according to the PIPE (PHY Interface for the PCI Express) standard. This standard has a lot of versions, the PIPE 3.0 version is used in this implementation, which supports up to PCIe Gen 2 speed. The interface is shown in Figure 17.The PHY is responsible for a lot of things, one of these important things is the detect sequence. At the detect state, following some electrical operations, when the i_PhyStatus signal becomes high, we check the value of i_RxStatus. If i_RxStatus = 000b, then there is no device detected at the other end of the link, else if i_RxStatus=011b, this indicates the detection of a device on the other end of the link.

2.1.2.2. Transmitter interface

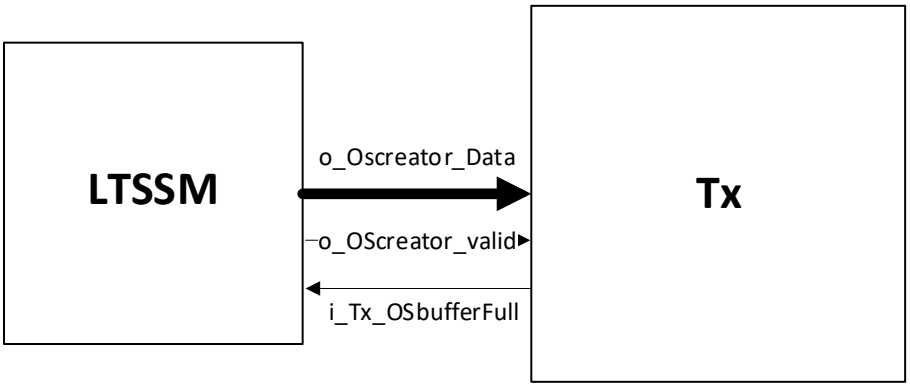

Figure 18: LTSSM-Tx interface

Figure 18 shows the interface between LTSSM and Tx. The interface between LTSSM and TX designed to achieve a reliable transmission of ordered sets through link. To achieve this reliable transmission, **we designed 2 signals which are**

- i_Tx_OSbufferFull: signal indicates if the transmitter buffer is full or not. If full the LTSSM needs to stop the transmission of order sets data.
- o_OScreator_valid: signal indicates if the data transmitted from LTSSM is a valid data or not.

The Table 6: LTSSM-Tx interface signals shows the values required for these signals to achieve a reliable negotiation of order sets.

Table 6: LTSSM-Tx interface signals

Signal	Direction	Description
i_Tx_OSbufferFull	Input	**Input from Transmitter block** • i_Tx_OSbufferFull =1, when Tx buffer is full • i_Tx_OSbufferFull =0, when Tx buffer not full
o_OScreator_Data[15:0]	Output	**Output to Transmitter block** Out the Data of the order set to transmitter block
o_OScreator_valid	Output	**Output to Transmitter block** Output signal to tell Tx that the data transmitted is valid ordered sets • o_OScreator_valid =1, valid order sets • o_OScreator_valid =0, not valid data

2.1.2.3. Receiver interface

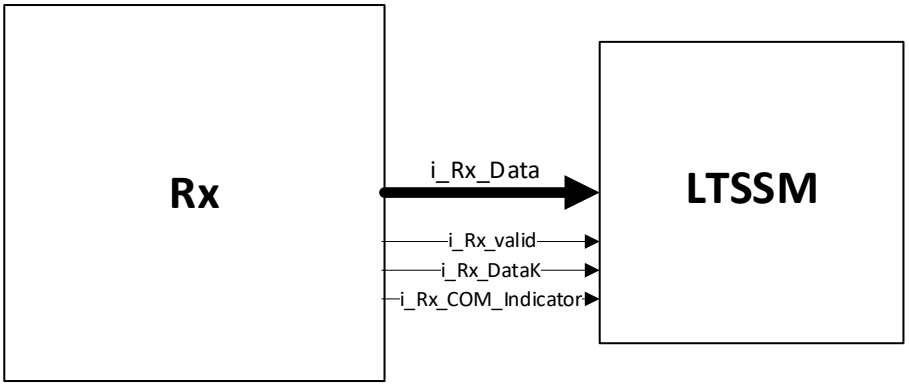

Figure 19: LTSSM-Rx interface

The interface between LTSSM and Rx designed to achieve a reliable negotiation of order sets. so, **we designed 3 signals which are**

- i_Rx_DataK: Data and control character indicator.
- i_Rx_valid: Valid signal coming from receiver.
- i_Rx_COM_Indicator: Indicator signal to indicate the beginning of order set

The next table shows the values required for these signals to achieve a reliable negotiation of order sets.

Table 7: LTSSM-Rx interface signals

Name	Direction	Description
i_Rx_Data[15:0]	Input	**Input from receiver** Data coming from receiver
i_Rx_DataK[1:0]	Input	**Input from receiver** Data and control character indicator • i_Rx_DataK=00, The 2 bytes are control character. • i_Rx_DataK=01, The least byte is data and the most byte is control character. • i_Rx_DataK=10, The least byte is control character and the most byte is data. • i_Rx_DataK=11, The 2 bytes are data.
i_Rx_valid[1:0]	Input	**Input from receiver** Valid signal coming from receiver • i_Rx_valid =00, the 2 bytes are not valid. • i_Rx_valid =01, the least byte is valid and the most is not valid. • i_Rx_valid =10, the least byte is not valid and the most is valid. • i_Rx_valid =11, the 2 bytes are valid
i_Rx_COM_Indicator[1:0]	Input	**Input from receiver** Indicator signal to indicate the beginning of order set • i_Rx_COM_Indicator =00, there is no COM character. • i_Rx_COM_Indicator =01, The least byte is COM and the most byte is not a COM. • i_Rx_COM_Indicator =10, The least byte is not a COM and the most byte is a COM. • i_Rx_COM_Indicator =11, invalid case.

2.1.3. LTSSM hardware description

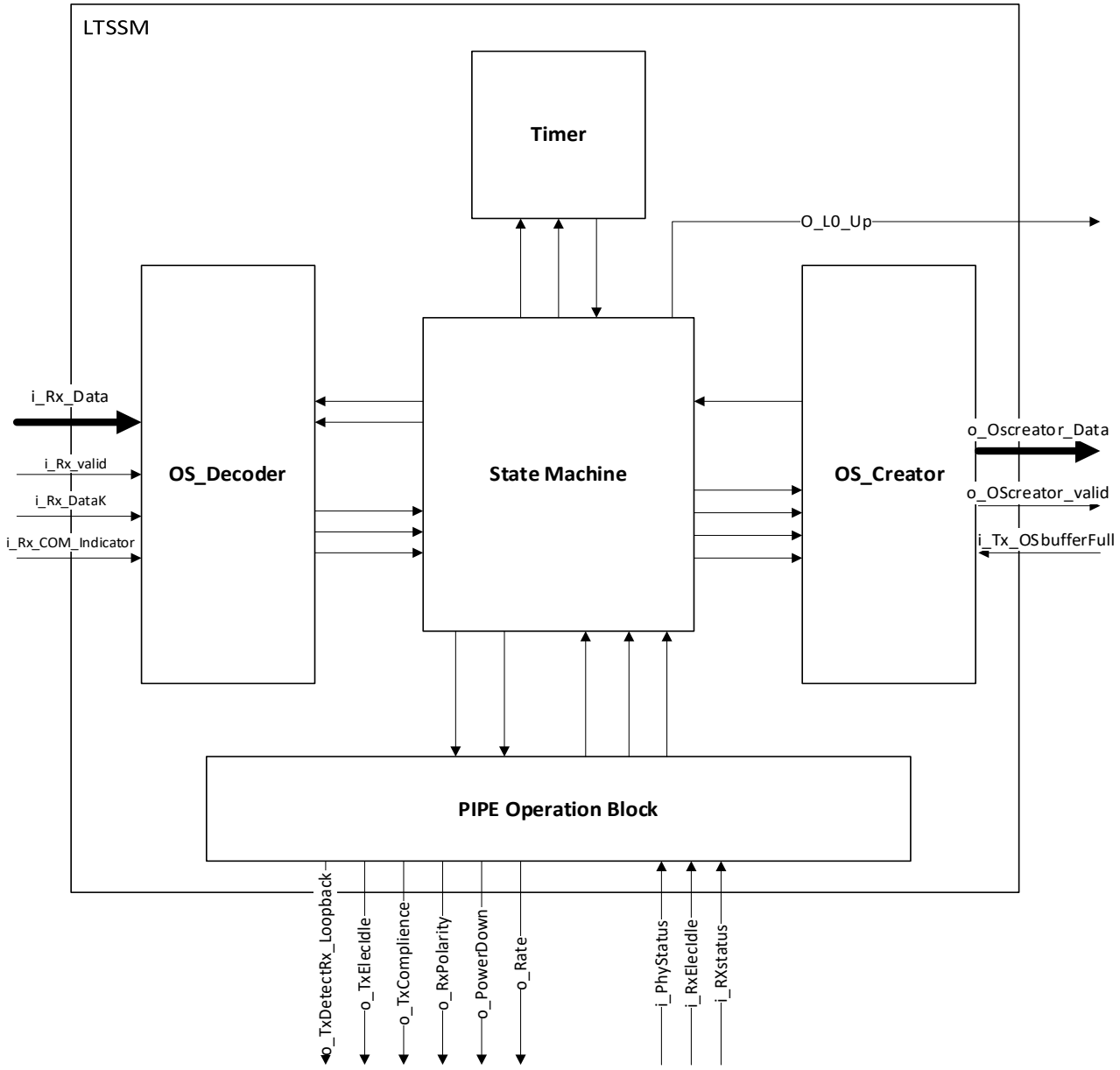

Figure 20: LTSSM implementation

Figure 20 shows the proposed design of LTSSM. LTSSM consists of 5 modules; the OS_Decoder that receive the data from the Rx, decodes it, the OS_Creator which forms the Ordered-Sets and moves it to the Tx to put it on the link. Both of these modules communicate with the LTSSM. StateMachine module which determine the link state whether it is a link training state or the full operation state L0. The timer module helps the StateMachine to set timeout values needed for different state (more details on this in the StateMachine module description). The PIPE Operation Block handles the interface between the LTSSM and the PHY according to the PIPE standard.

Detailed description of each module is presented in this subsection. Note that CLK and RST signal are not explicitly drawn for sequential modules, and all the input and output signal are explicitly mentioned in each module abstracted diagram, but not all of them mentioned in the detailed diagram for simplicity.

2.1.3.1. OS_Creator module

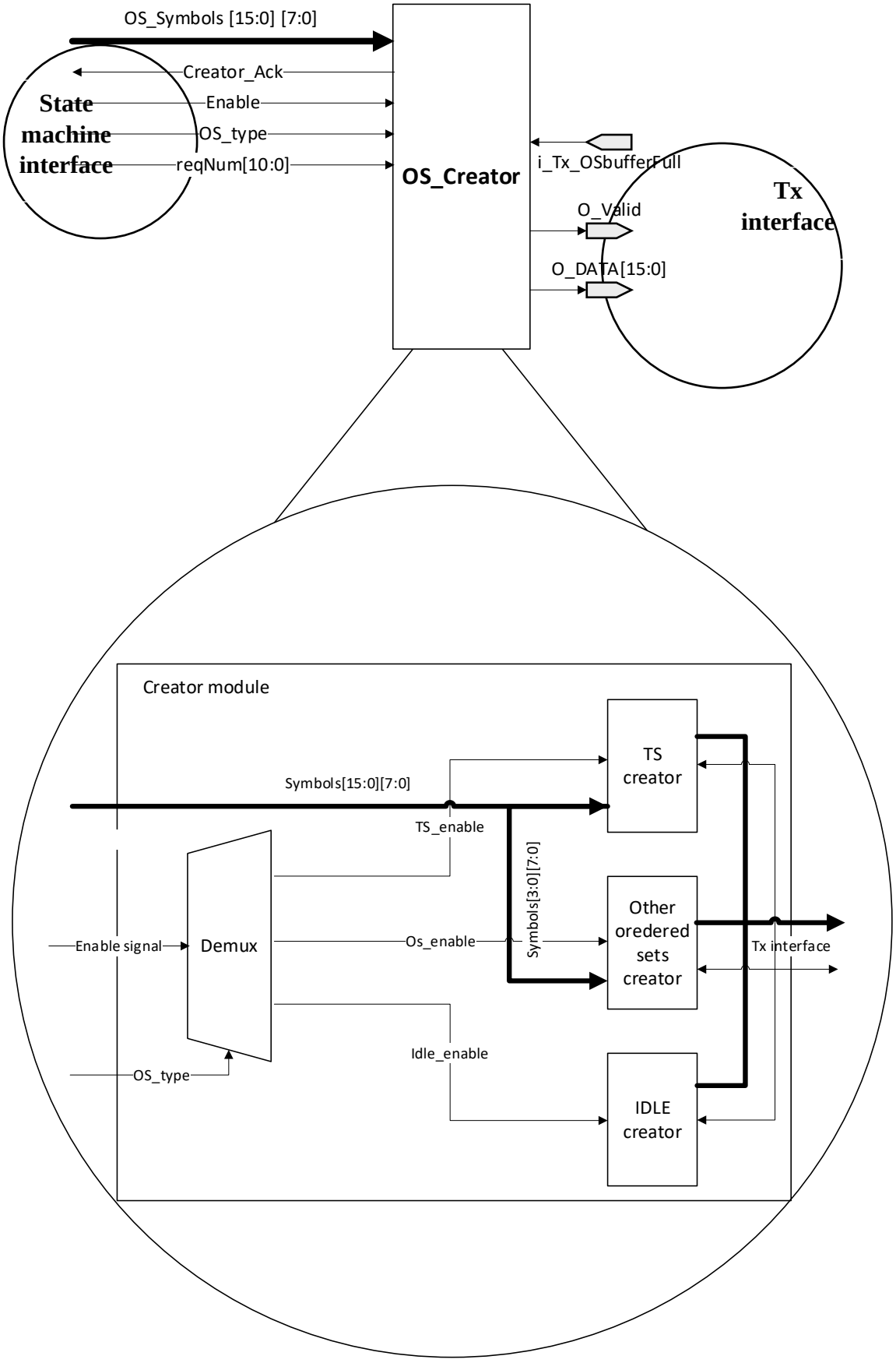

Figure 21: OS_Creator implementation

OS_Creator module main functionality

This module is responsible for the creation of the required Ordered-Sets to complete the training process. This module represents the LTSSM interface with the Tx block. As shown in

Figure 21, The creator module composed of 3 sub-modules, one for the creation of Training Sequence TS ordered sets, the second one for the creation of the other ordered sets (SKP, EIEOS, and IDLE) and the last one for the creation of the logical idle (00h) symbols.

Theory of Operation

According to its current state, the LTSSM StateMachine asserts the enable signal of the creator, sets the type of Ordered-Sets to be formed using OS_type signal, and sends the required symbols of the Ordered-Sets to be formed along with the required number of these Ordered-Sets to be transmitted through reqNum. upon finishing the OS transmission, OS_Creator issues an acknowledgment signal through creator_Ack signal back to the LTSSM StateMachine.

Inputs and outputs

Table 8: OS_Creator signals

Signal	Direction	Description
i_Tx_OSbufferFull	Input	As described in interface with transmitter section
o_OScreator_Data[15:0]	Output	
o_OScreator_valid	Output	
OS_Symbols [15:0][7:0]	Input	**Input from StateMachine** Data of Order sets coming from State machine
Enable	Input	**Input from StateMachine** Enable signal to the module.
OS_type [1:0]	Input	**Input from StateMachine** Choose between TS creator or other order set creator • OS_type=00, Other order set creator • OS_type=01, Choose TS creator • OS_type=10, Choose logical idle creator
reqNum [10:0]	Input	**Input from StateMachine** Number of repetitions of the OS.
Counter_Ack	Output	**Output to StateMachine** • States that the number of the transmitted OS is equal to or more than the reqNum.

2.1.3.2. State machine module

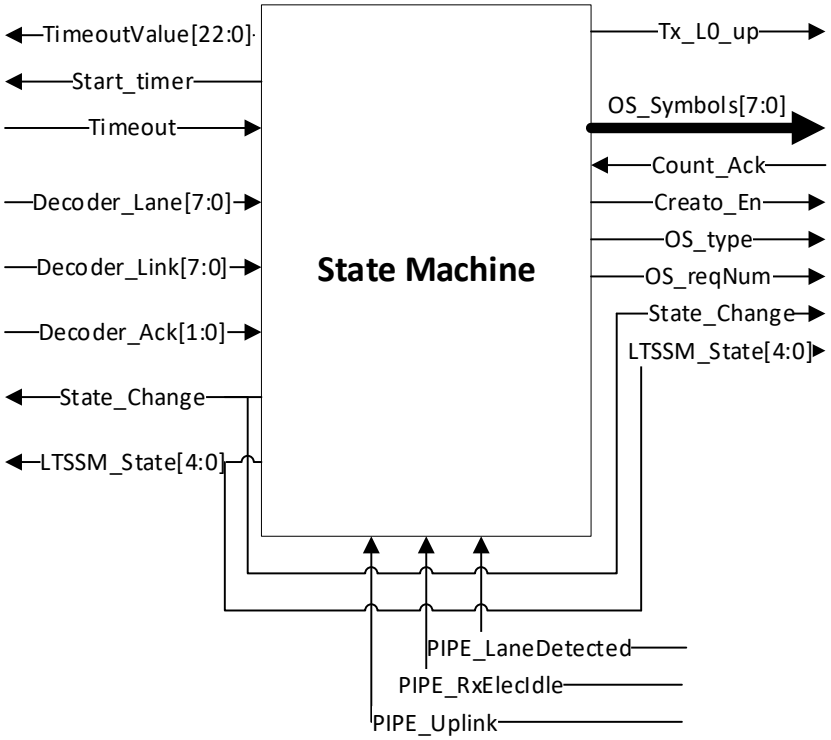

Figure 22: StateMachine module

State machine module main functionality

- Take the required signals from PIPE and Decoder for controlling the transitions between the states in the State machine module.
- Out the number of TS need to be sent to Order set creator block.
- Out the LTSSM states to PIPE interface, and get back the signals required for controlling the transitions between the states in the LTSSM block.
- After finishing the LTSSM training, L0_up signal indicates the normal operation.

The State diagrams

Figure 23 shows us the states of the link training and status state machine (LTSSM), each state consists of substates, the LTSSM consists mainly of 11 top-level states: Detect, Polling, Configuration, Recovery, L0, L0s, L1, L2, hot reset, Loopback and Disable. but in our design, we decide to implement only 5 states: Detect, Polling, Configuration, Recovery and L0. The normal flow of link training is Detect → Polling → Configuration → L0.

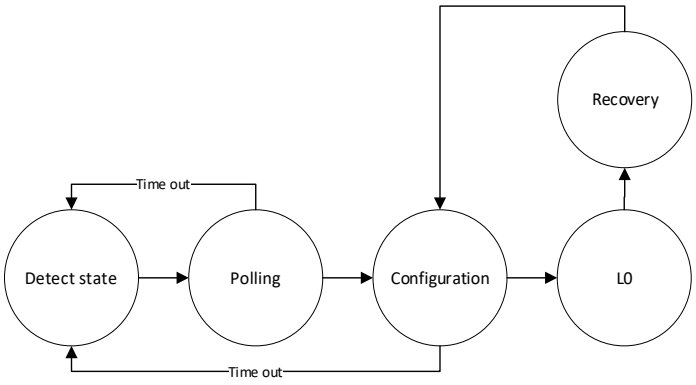

Figure 23: LTSSM states

Detect state description

The detect state is the first state after the reset, in this state a device is electrically detects a receiver. Also, we can enter this state from the polling and configuration state. The detect state composed of two substates shown in Figure 24: detect quiet and detect active.

- **Detect quiet** is the initial substate, in this substate the transmitter in electrical idle, the intendent data rate 2.5 GT/s (Gen1 speed) and the Linkup=0 which mean the link is not operational.
- **Detect active,** this substate entered from the detect quiet, in this substate the transmitter tests if the receiver is connected or no. the next substate after the detect active, when receiver detected is the polling active substate.

✓ Note: the required condition for the transition between substate to another shown in Table 9

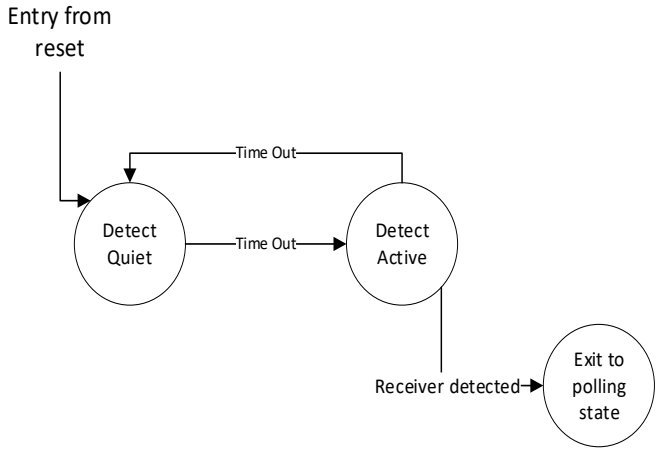

Figure 24: detect state description

Polling state description

The polling state enters from the detect active after the receiver detection, in this state the transmitter starts to send TS1 and TS2.The goal of this state is to achieve the bit lock, symbol lock and learn the available lane data rate. This state composed of three substates but we implement only the first two: polling active and polling configuration.

- **Polling active:** in this substate, transmitter sends a minimum number of TS1s with pad lane and link number on all detected lanes.
- **Polling configuration:** in this substate, transmitter will stop sending TS1s and start sending TS2s with lane and link number (the purpose of sending TS2s is to advertise the link partner that this device is ready to proceed to the next state).

✓ Note: the required condition for the transition between substate to another shown in Table 9Table 9

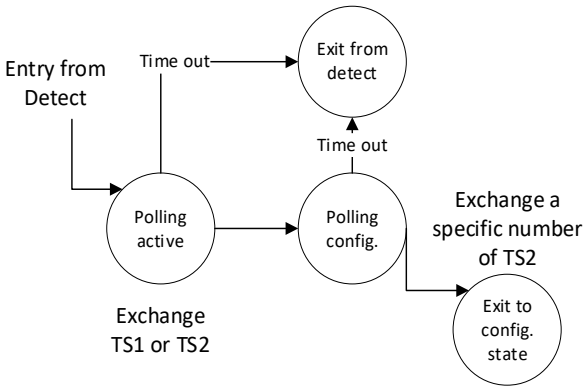

Figure 25: Polling state description

Configuration state description

The configuration state enters from polling configuration, or from the recovery state in this state the upstream and the downstream continues exchange TS1s and TS2s. the goal of this state is to determine the link width and to assign the lane numbers. This state composed of 6 substates: configuration linkwidth start, configuration linkwidth accept, configuration lanenum wait, configuration lanenum accept, configuration complete and configuration idle. In this state the downstream device differs from upstream device.

- **Configuration linkwidth start:**

  Downstream device: The downstream now is the leader device, will initiate a link number and starts to sends TS1s with a non-PAD link number while the lane number remains PAD.
  Upstream device: The upstream now is the follower and goes back to sending TS1s with link and lane PAD.
- **Configuration linkwidth accept:**
  Downstream device: The downstream now, will initiate a lane number and starts to sends TS1s with a non-PAD link number and non-Pad Lane number.
  Upstream device: The upstream now sent back TS1s with one of the link number selected and PAD lane number.
- **Configuration lanenum wait**:
  Downstream device: The downstream will continue to sends TS1s with a non-PAD link number and non-Pad Lane number.
  Upstream device: The upstream will sends TS1s with a non-PAD link number and non-Pad Lane number.
- **Configuration lanenum Accept**:
  Downstream device: The downstream will continue to sends TS1s with a non-PAD link number and non-Pad Lane number.
  Upstream device: The upstream will sends TS1s with a non-PAD link number and non-Pad Lane number
- **Configuration complete**:
  Downstream device: The downstream starts to sends TS2s with a non-PAD link number and non-Pad Lane number.
  Upstream device: The upstream will sends TS2s with a non-PAD link number and non-Pad Lane number.

- **Configuration idle**: during this substate the devices sends idle data and wait for minimum number of idle data after receiving it the link can transition to L0, where the LinkUp=1.

✓ Note: the required condition for the transition between from substate to another shown in Table 10

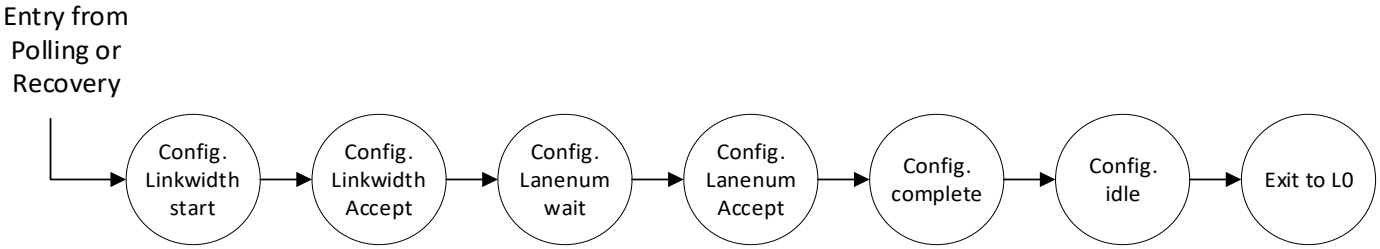

Figure 26: configuration state description

L0 state description

L0 state is the normal full-operational link state, during which Logical idle, TLPs and DLLPs are exchange between links. The physical layer notifies the upper layers that the link is ready for operation by setting LinkUp=1.

Note: the required condition for the transition between substate to another shown in Table 11

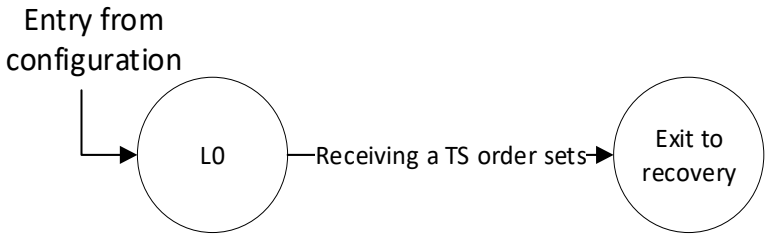

Figure 27: L0 state

Recovery state description

We design a small recovery state, to keep operating with Mentor (Siemens EDA) IP when Mentor IP enters the recovery state for any reasons, we can complete our link training.

LTSSM states exit conditions

According to the PCI-e standard, the exit condition for each state is as follows

Table 9: LTSSM state exit conditions (Detect and Polling)

States	Substates	Condition
Detect	**Detect.Quiet to Detect.Active**	12 ms timeout or lane exit electrical idle
	Detect.Acive to Detect.Quiet	12 ms timeout, no lane detected
	Detect.Acive to Polling	Lane detected

Polling	**Polling.Active to Detect.Quiet**	24 ms timeout
	Polling.Active to Polling.Config	1024 TS1 sent with training control =0, 8 received **Or** 1024 TS1 sent with training control =4, 8 received **Or** 1024 TS2 sent, 8 received
	Polling.Config to Detect.Quiet	48 ms timeout
	Polling.Config to Configration	16 TS1 sent, 8 received

Starting from configuration state the exit condition will differ from downstream device Or Upstream device.

Table 10: Configuration state exit conditions

UP/Down		**Exit condition**
Downstream	**Config.Linkwidth.start to Config.Linkwidth.Accept**	Received 2 TS1 with non pad link number
	Config.Linkwidth.Accept to Config.Lanenum.wait	Received 2 TS1 with the link saved and Pad lane
	Config.Lanenum.wait to Config.Lanenum.Accept	Received 2 TS1 with the link saved and non pad lane
	Config.Lanenum.Accept to Config.Complete	Received 2 TS1 with the link saved and lane saved
Upstream	**Config.Linkwidth.start to Config.Linkwidth.Accept**	Received 2 TS1 with non pad link number
	Config.Linkwidth.Accept to Config.Lanenum.wait	Received 2 TS1 with the link saved and non Pad lane
	Config.Lanenum.wait to Config.Lanenum.Accept	Received 2 TS2 with the link saved and Lane saved
	Config.Lanenum.Accept to Config.Complete	Received 2 TS2 with the link saved and Lane saved

Starting from L0 state

Table 11: L0 state exit condition

States	**Substates**	**Condition**
L0	**L0 to recovery**	Received a TS order sets

Inputs and outputs

Table 12: State Machine signals

Name	Direction	Description
OS_Symbols [15:0][7:0]	Output	**Output to creator** The order set data
OS_type	Output	**Output to creator** Choose between TS creator or other order set creator • Types=1, Choose TS creator • Types=0, Other order set creator
OS_reqNum	Output	**Output to creator** Number of repetitions of the order set
Creator_En	Output	**Output to creator** To enable the creator to send the order set
Counter_Ack	Input	**Input from creator** To tell the State Machine the required number of ordered sets had been transmitted • Counter_Ack = 1, Transmitted OS ≥ OS_reqNum • Counter_Ack = 0, otherwise.
LTSSM_UpLink	Input	**Input from PIPE interface** Indicates bit and symbol lock, ready to L0 state
LTSSM_RxElecidle	Input	**Input from PIPE interface** Mirror the i_RxElecIdle coming from PHY to LTSSM
LTSSM_LaneDetected	Input	**Input from PIPE interface** Indicate successful detection of a receiver to LTSSM
LTSSM_state[4:0]	Output	**Output to PIPE interface & decoder** To update the current state
Decoder_Ack[1:0]	Input	**Input from decoder** Acknowledgement signal on the number of TS order sets received to Control the transition between the states.
Decoder_Lane[7:0]	Input	**Input from decoder** The lane number decoded from TS1 and TS2
Decoder_Link[7:0]	Input	**Input from decoder** The link number decoded from TS1 and TS2
State_change	Output	**Output to decoder** to update the decoder by the change in the state
Timeout value[22:0]	Output	**Output to timer** Time needed by timer to timeout

Start	Output	**Output to timer** To start count the time • Start=1, Start count • Start=0, Stop count and restart all the time data (when we start again, we will start from 0)
Timeout	Input	**Input from timer** • Time Out =1, time ≥ Timeout value • Time Out=0, otherwise.
L0_UP	Output	**Output to Transmitter block** To give the transmitter block the permission of working in normal operation. • L0_UP=1, indicate the L0 state

2.1.3.3. Timer module

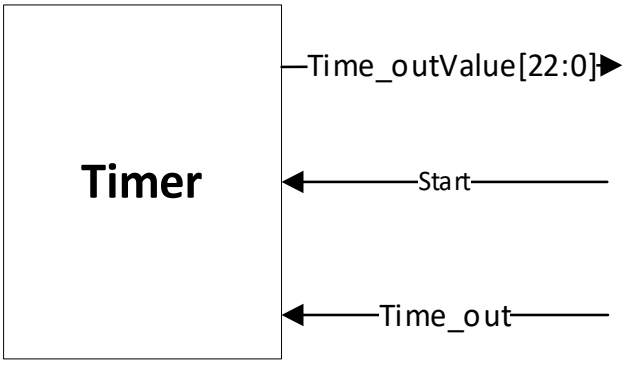

Figure 28: Timer module

Timer module main functionality

The main function of timer is to provide the time out signal to the StateMachine according to the given timeout value.

Inputs and outputs

Table 13: Timer signal

Name	Direction	Description
Time out value [22:0]	Input	**Input from State machine** Time, we need to reach
Start	Input	**Input from State machine** To start count the time • Start=1, Start count • Start=0, Stop count and restart all the time data (when we start again, we will start from 0)

Time Out	Output	**Output to State machine** • Time Out =1, when we reach the time needed • Time Out=0, when we still count the time needed

2.1.3.4. Decoder module

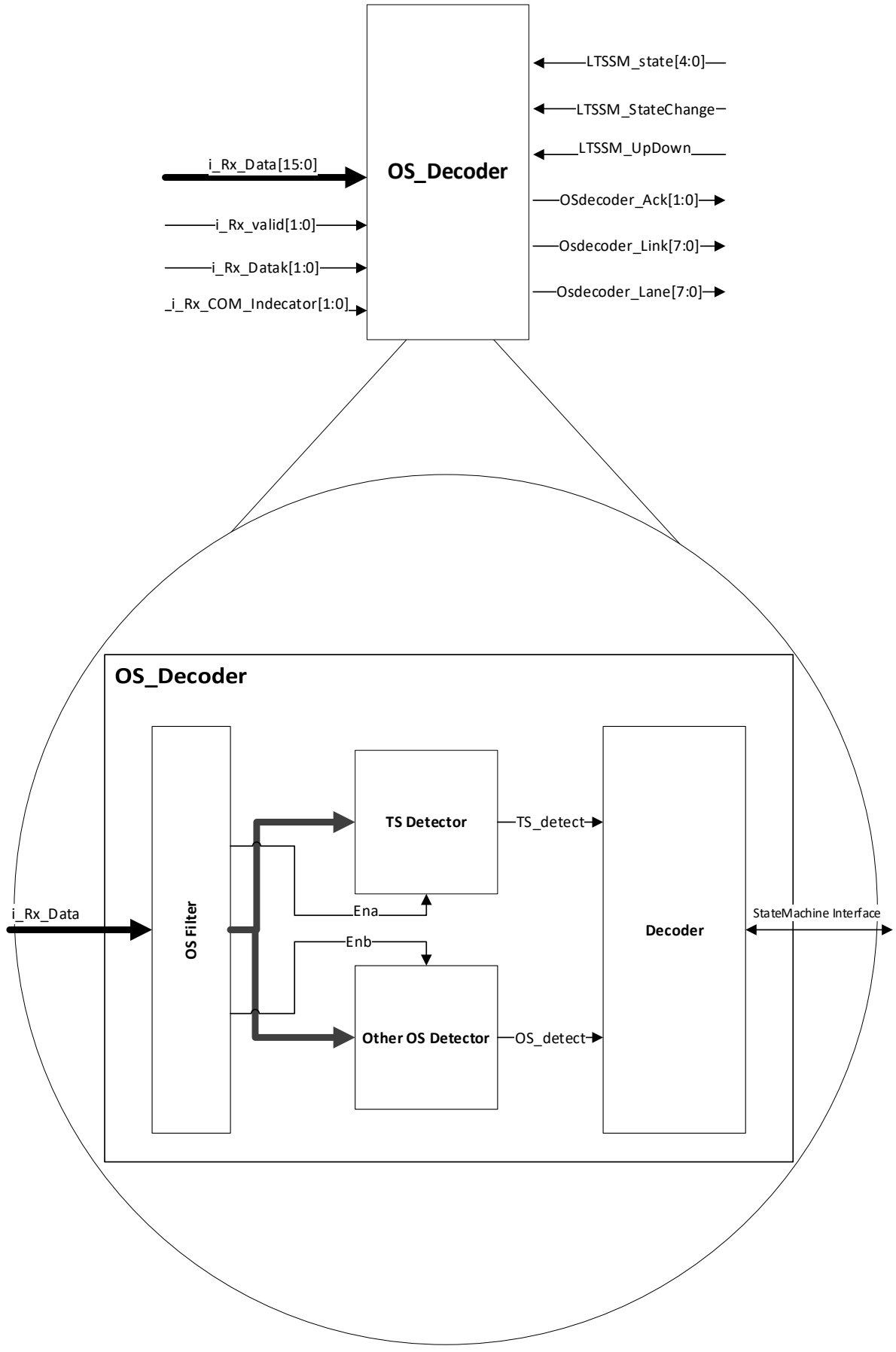

Figure 29: OS_Decoder implementation

Decoder main functionality

The Decoder block receives, counts, and decodes the data coming from the PCIe Rx along with some additional signals. It consists of four modules, the OS_Filter, the TS_Detector, the OtherOS_Detector, and the Decoder_MainBlock as shown in Figure 29.

Theory of operation

The filter, check if the received OS is a TS or an OtherOS, it passes it to the TS_Detector or OtherOS_Detector to checks the sanity of the OS, then the detector issues a detect signal to the Decoder_MainBlock, which checks the components of the OS and compare it with the expected value from the protocol according to each state, if they match it adds it to some counter waiting it to reach the required value of OS in this state. When the counter reaches this value, it rises an acknowledgment signal to the StateMachine.

Inputs and outputs

Table 14: OS_Decoder signals

Name	Direction	Description
i_Rx_Data[15:0]	Input	**Data coming from receiver**
i_Rx_DataK[1:0]	Input	Data and control character indicator • i_Rx_DataK=00, The 2 bytes are control character. • i_Rx_DataK =01, The least byte is data and the most byte is control character. • i_Rx_DataK =10, The least byte is control character and the most byte is data. • i_Rx_DataK =11, The 2 bytes are data.
i_Rx_valid[1:0]	Input	**Valid signal coming from receiver** • i_Rx_valid =00, the 2 bytes are not valid. • i_Rx_valid =01, the least byte is valid and the most is not valid. • i_Rx_valid =10, the least byte is not valid and the most is valid. • i_Rx_valid =11, the 2 bytes are valid
i_Rx_COM_Indicator[1:0]	Input	Indicator signal to indicate the beginning of ordered set • i_Rx_COM_Indicator =00, there is no COM character. • i_Rx_COM_Indicator =01, The least byte is COM and the most byte is not a COM. • i_Rx_COM_Indicator =10, The least byte is not a COM and the most byte is a COM. • i_Rx_COM_Indicator =11, invalid case.
OSdecoder_Ack[1:0]	Output	Acknowledgement signal on the number of TS receiver to Control the transition between the states.
OSdecoder_Lane[7:0]	Output	The lane number decoded from TS1 and TS2
OSdecoder_Link[7:0]	Output	The link number decoded from TS1 and TS2

2.1.3.5. Pipe interface module

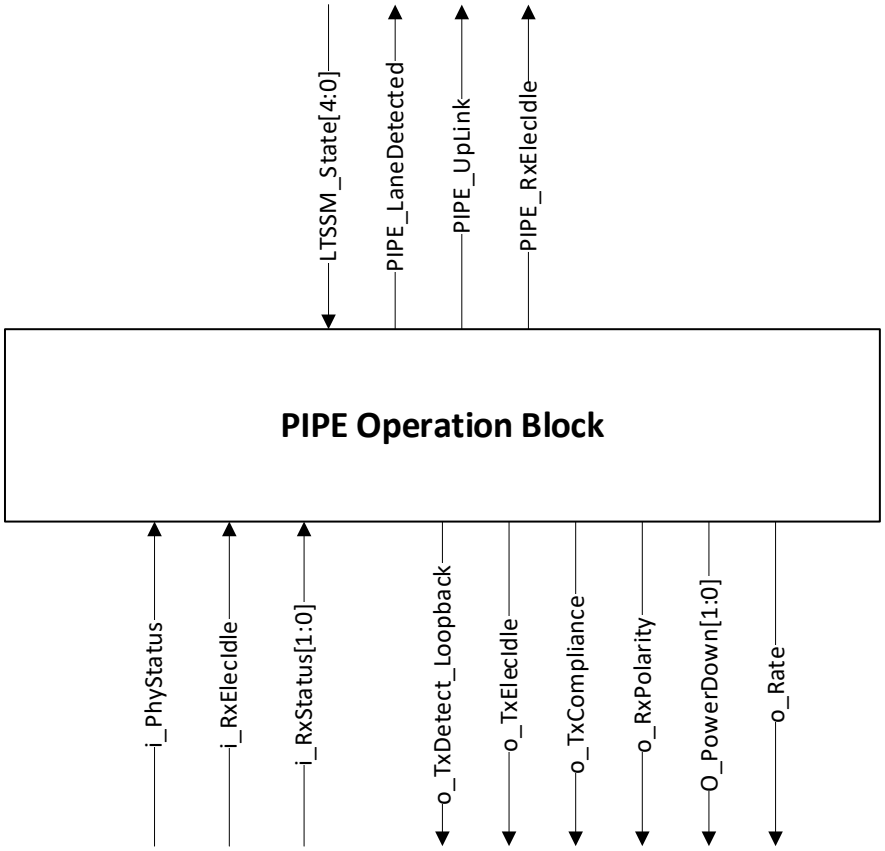

Figure 30: PIPE operation module

Pipe interface module main functionality

- The main interface between the LTSSM and the PHY layer according to the PIPE standard.
- It is used in performing the detect sequence of a receiver connected to the device's transmitter.

Theory of operation

It takes the state of LTSSM ***i_LTSSM_State*** as an input and determine the value of the input signals to PHY, indicating a specific operation-like detect sequence and decode the PHY output signals to indicate a certain result, upon it the LTSSM takes a certain action.

Inputs and outputs

Note: (*) means that this signal coming from PIPE standard.

Table 15: PIPE operation signals

<table>
<tr><th>Signal</th><th>Direction</th><th>Description</th></tr>
<tr><td>LTSSM_State [4:0]</td><td>Input</td><td>Inputs from State machine
Indicate the current state</td></tr>
<tr><td>i_PhyStatus*</td><td>Input</td><td>Inputs from PHY
Used to communicate completion of several PHY functions including stable PCLK after Reset_n deassertion, power management state transitions, rate change, and receiver detection.</td></tr>
<tr><td>i_RxElecIdle*</td><td>Input</td><td>Inputs from PHY
Indicates receiver detection of an electrical idle.</td></tr>
<tr><td>i_RxStatus* [2:0]</td><td>Input</td><td>Inputs from PHY
Encodes receiver status and error codes for the received data stream when receiving data.
<table>
<tr><td>Value</td><td>Description</td></tr>
<tr><td>000</td><td>Received data OK</td></tr>
<tr><td>001</td><td>1 SKP added</td></tr>
<tr><td>010</td><td>A SKP removed</td></tr>
<tr><td>011</td><td>Receiver detected</td></tr>
<tr><td>100</td><td>Both 8B/10B decode error and receive disparity error</td></tr>
<tr><td>101</td><td>Elastic buffer overflow</td></tr>
<tr><td>110</td><td>Elastic buffer under flow</td></tr>
<tr><td>111</td><td>Receive disparity error</td></tr>
</table></td></tr>
<tr><td>o_TxDetectRx_loopback*</td><td>Output</td><td>Output to PHY
Used to tell the PHY to begin a receiver detection operation or to begin loopback</td></tr>
<tr><td>o_TxElecIdle*</td><td>Output</td><td>Output to PHY
Tells PHY that the transmitter is electrically idle</td></tr>
<tr><td>o_TxCompliance*</td><td>Output</td><td>Output to PHY
Sets the running disparity to negative. Used when transmitting the PCI Express compliance pattern</td></tr>
<tr><td>o_RxPolarity*</td><td>Output</td><td>Output to PHY
Tells PHY to do a polarity inversion on the received data.</td></tr>
<tr><td>o_PowerDown* [1:0]</td><td>Output</td><td>Output to PHY
Power up or down the transceiver
<table>
<tr><td>Value</td><td>Description</td></tr>
<tr><td>00</td><td>L0, normal operation</td></tr>
<tr><td>01</td><td>L0s, low recovery power saving state</td></tr>
<tr><td>10</td><td>L1, long recovery power saving state</td></tr>
<tr><td>11</td><td>L2, lowest power state</td></tr>
</table></td></tr>
<tr><td>o_Rate*</td><td>Output</td><td>Output to PHY
Control the link signaling rate.</td></tr>
<tr><td>PIPE_UpLink</td><td>Output</td><td>Output to LTSSM
Indicates bit and symbol lock, ready to L0 state</td></tr>
</table>

PIPE_RxElecidle	Output	**Output to LTSSM** Mirror the i_RxElecIdle coming from PHY to LTSSM
PIPE_LaneDetected	Output	**Output to LTSSM** Indicate successful detection of a receiver to LTSSM

2.2. Tx

2.2.1. Tx Overview

The functions of the Tx block can be summarized in the following points:

1. Receive the packets from the data link layer either it is TLP or DLLP.
2. Capsulate the TLP and DLLP.
3. Receive the ordered set packets from the LTSSM.
4. Handle the priority between the packets coming from the data link layer and the packets from LTSSM.
5. If there is no data from the data link layer and LTSSM, it should send logical idle on the lane.

Figure 31 shows the block diagram of the Tx block.

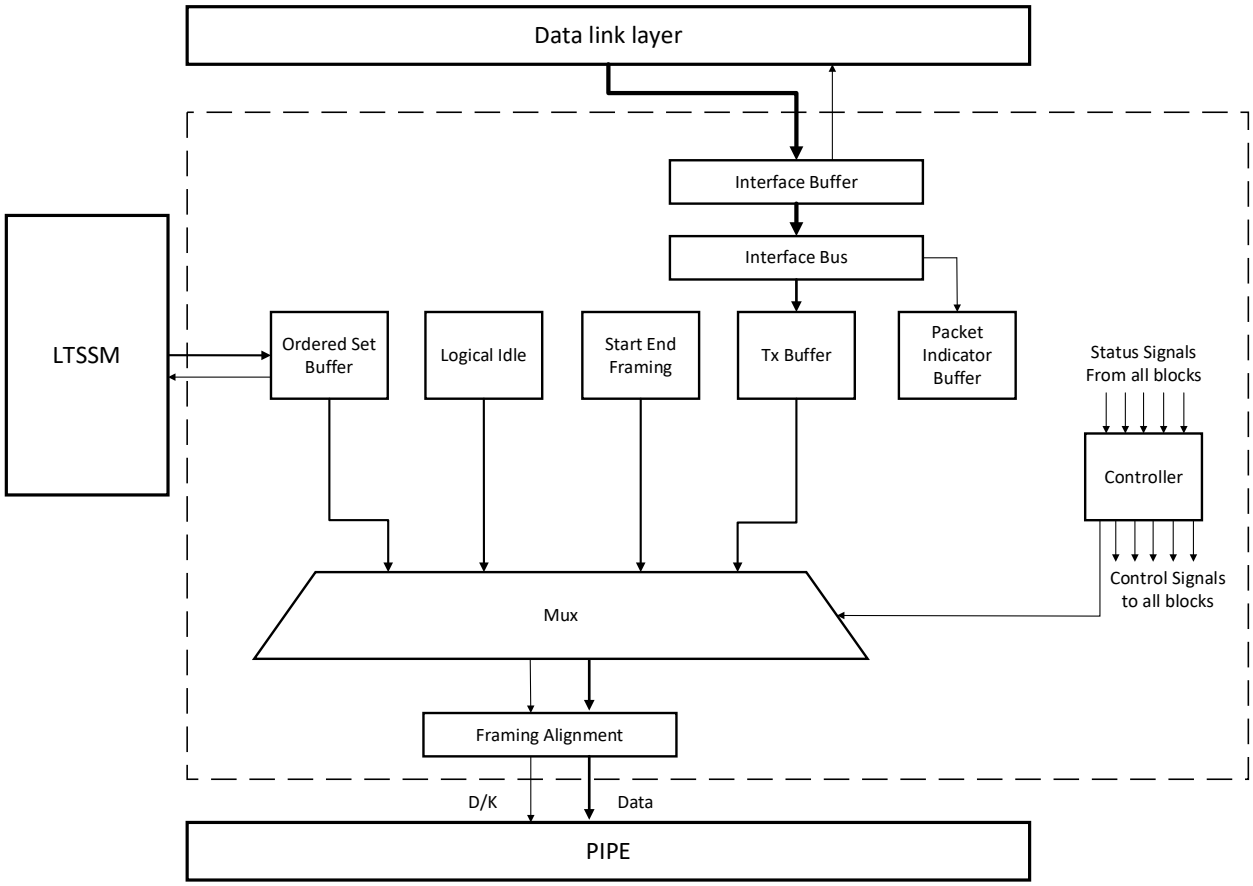

Figure 31: Tx - Block diagram

2.2.2. Tx interface with PIPE & LTSSM

The block diagram of the Tx top module is shown in Figure 32

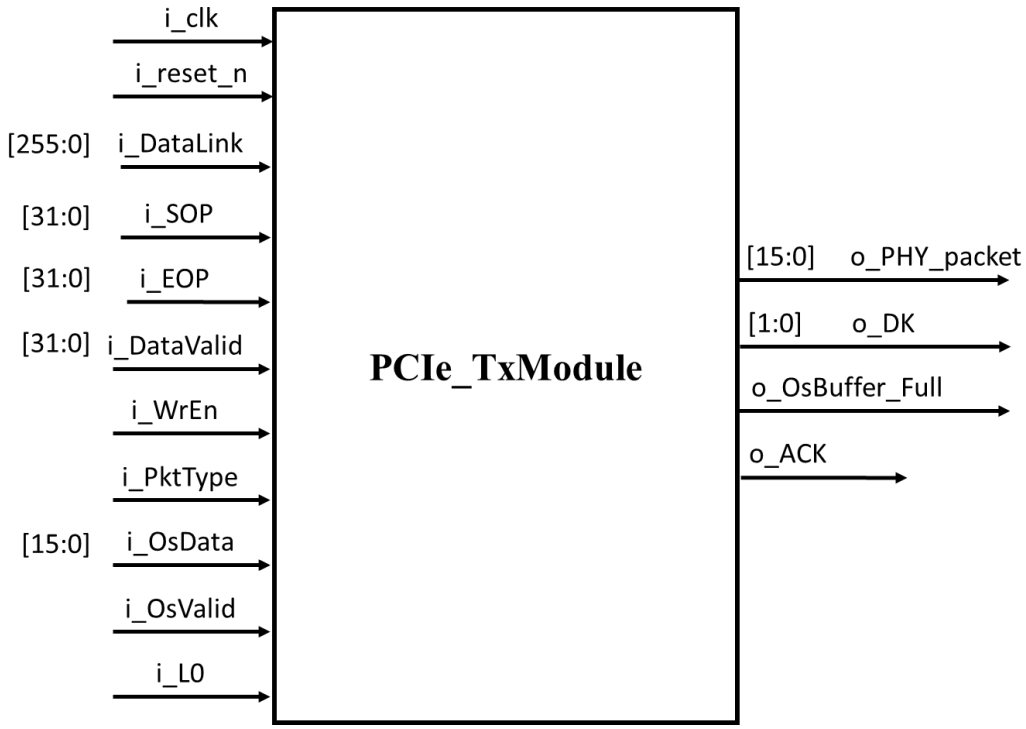

Figure 32: Tx - Top Module - Block diagram

2.2.2.1. Interface with the data link layer

Table 16 illustrates the signals of the interface between the Tx and the data link layer.

Table 16: Tx-Data Link Layer interface signals

Name	Direction	Size	Description
i_DataLink	Input	256 bits	**Input from the data link layer** The data which comes from the data link layer. If the packet is more than 32 Bytes, each 32 bytes of it comes in a single clock cycle.
i_SOP	Input	32 bits	**Input from the data link layer** The start of packet indicator. Each bit refers to the corresponding byte. The bit which equals 1 indicates to the start byte of the packet.
i_EOP	Input	32 bits	**Input from the data link layer** The end of packet indicator. Each bit refers to the corresponding byte. The bit which equals 1 indicates to the end byte of the packet.
i_DataValid	Input	32 bits	**Input from the data link layer** The valid indicator. Each bit refers to the corresponding byte. All bytes between the start byte and the end byte must be 1.
i_WrEn	Input	1 bit	**Input from the data link layer** Write enable of the buffer. When it is 1, the data link layer inserts the packet in the buffer.

i_PktType	Input	1 bit	**Input from the data link layer** It determines whether the packet is TLP or DLLP. 0 for TLP and 1 for DLLP.
o_ACK	Output	1 bit	**Output to the data link layer** When the interface buffer receives the end of packet, it raises this signal to prevent the data link layer from sending any other packet. When this signal is high, the interface bus can read the data from this buffer. On the other hand, when this signal is low, the data link layer can send packets to the buffer and the interface bus can't read the data from the buffer.

2.2.2.2. Interface with the LTSSM

Table 17 illustrates the signals of the interface between the Tx and the LTSSM.

Table 17: Tx-LTSSM interface signals

Name	Direction	Size	Description
i_OsData	Input	16 bits	**Input from the LTSSM** Two bytes of the ordered set packet.
i_OsValid	Input	1 bit	**Input from the LTSSM** It acts as the write enable of the buffer. When it is 1, the LTSSM inserts two bytes in the buffer.
i_L0	Input	1 bit	**Input from the LTSSM** If it equals 1, the normal operation mode is activated. If it equals 0, LTSSM is still in link training or initialization mode.
o_OsBuffer_Full	Output	1 bit	**Output to the LTSSM** Full flag to indicate that the buffer is full. So, the LTSSM can't send any new packet.

2.2.2.3. Interface with the PIPE

Table 18 illustrates the signals of the interface between the Tx and the PIPE.

Table 18: Tx-PIPE interface signals

Name	Direction	Size	Description
o_PHY_packet	Output	16 bits	**Output to the PIPE**
o_DK	Output	2 bits	**Output to the PIPE**

2.2.3. Tx hardware description

2.2.3.1. Data link layer – MAC layer Interface Buffer

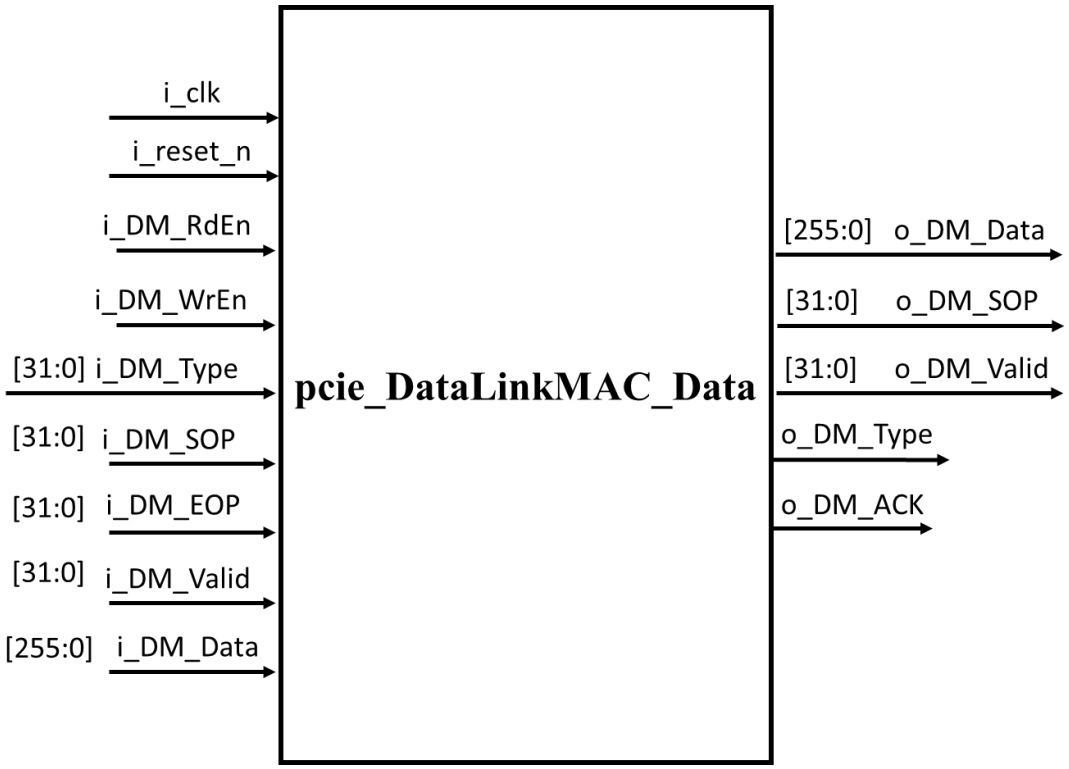

Figure 33: Interface Buffer module

Block main functionality:

This module is the interface between the data link layer and MAC layer. It is used to receive a complete single packet at a time from the data link layer then send it raw by raw to the "Data link layer – MAC layer Interface Bus". As long as this module doesn't send the full packet to the interface bus, it can't receive any other packet from the data link layer. *ACK* signal which is an output signal from this module controls this process. This signal goes to the data link layer and the interface bus.

The maximum allowable size of TLP is 544 Bytes. So, this module is designed as a FIFO with 17 rows depth and 32 Bytes width. It takes status signals for each row to indicate the length of each packet. These signals are start of packet, end of packet and valid. Each bit of these signals refers to each byte the corresponding row. There are some restrictions on these signals. For instance, the start of the packet should be the first byte in the row, and the valid discontinuity is not allowed which means that all bytes between the start of packet byte and the end of packet byte must be valid.

The block diagram of the interface buffer is shown in Figure 33. Table 19 illustrates the signals of the interface buffer.

Table 19: Interface_Buffer Signals

Name	Direction	Size	Description
i_clk	Input	1 bit	Clock signal for positive edge registers

i_reset_n	Input	1 bit	Global asynchronous reset.
i_DM_RdEn	Input	1 bit	**Input from the interface bus** Read enable of the buffer. When it is 1, the buffer outs a row from its data to the interface bus.
i_DM_WrEn	Input	1 bit	**Input from the data link layer** Write enable of the buffer. When it is 1, the data link layer inserts the packet in the buffer.
i_DM_Type	Input	1 bit	**Input from the data link layer** It determines whether the packet is TLP or DLLP. 0 for TLP and 1 for DLLP.
i_DM_SOP	Input	32 bits	**Input from the data link layer** The start of packet indicator. Each bit refers to the corresponding byte. The bit which equals 1 indicates to the start byte of the packet.
i_DM_EOP	Input	32 bits	**Input from the data link layer** The end of packet indicator. Each bit refers to the corresponding byte. The bit which equals 1 indicates to the end byte of the packet.
i_DM_Valid	Input	32 bits	**Input from the data link layer** The valid indicator. Each bit refers to the corresponding byte. All bytes between the start byte and the end byte must be 1.
i_DM_Data	Input	256 bits	**Input from the data link layer** The data which comes from the data link layer. If the packet is more than 32 Bytes, each 32 bytes of it comes in a single clock cycle.
o_DM_Type	Output	1 bit	**Output to the interface bus** It determines whether the packet is TLP or DLLP. 0 for TLP and 1 for DLLP.
o_DM_SOP	Output	32 bits	**Output to the interface bus** The start of packet indicator. Each bit refers to the corresponding byte. The bit which equals 1 indicates to the start byte of the packet.
o_DM_Data	Output	256 bits	**Output to the interface bus** The data which is sent to the interface bus.
o_DM_Valid	Output	32 bits	**Output to the interface bus** The valid indicator. Each bit refers to the corresponding byte.
o_DM_ACK	Output	1 bit	**Output to the interface bus and the data link layer** When the interface buffer receives the end of packet, it raises this signal to prevent the data link layer from sending any other packet. When this signal is high, the interface bus can read the data from this buffer. On the other hand, when this signal is low, the data link layer can send packets to the buffer and the interface bus can’t read the data from the buffer.

2.2.3.2. Data link layer – MAC layer Interface Bus

Block main functionality:

This module is a single row of the interface bus with 32 Bytes width. Its function is to receive a packet row by row from the interface buffer and receive the status signals of the corresponding row to send the packet two bytes by two bytes to the Tx buffer and to send the packet indicators signals to the packet indicator buffer.

The block diagram of the interface bus is shown in Figure 34.

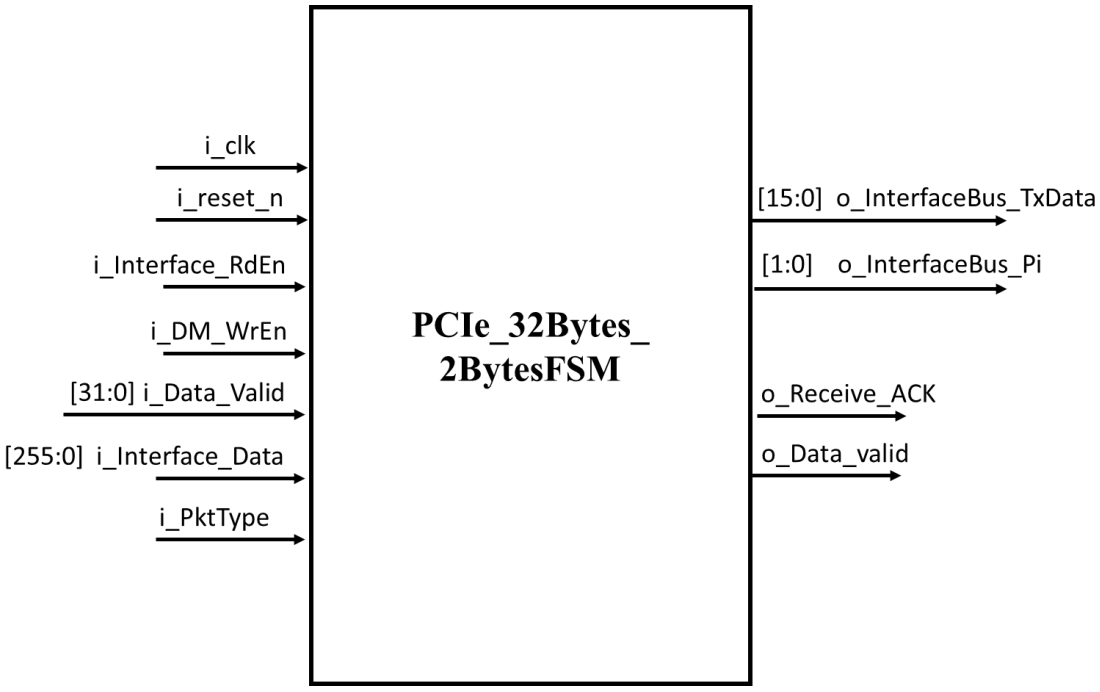

Figure 34: Interface Bus module

Table 20 illustrates the signals of the interface bus.

Table 20: Interface_Bus Signals

Name	Direction	Size	Description
i_clk	Input	1 bit	Clock signal for positive edge registers
i_reset_n	Input	1 bit	Global asynchronous reset.
i_Interface_RdEn	Input	1 bit	**Input from the Tx Buffer** Indicating whether the Tx buffer is full or not. When it is 1, the bus outs two bytes from its data to the Tx buffer.
i_Data_Valid	Input	32 bits	**Input from the interface buffer** The valid indicator. Each bit refers to the corresponding byte.
i_Interface_Data	Input	256 bits	**Input from the interface buffer** The data which is sent from the interface bus.
i_PktType	Input	1 bit	**Input from the interface buffer**

			It determines whether the packet is TLP or DLLP. 0 for TLP and 1 for DLLP.
o_InterfaceBus_TxData	Output	16 bits	**Output to the Tx buffer** The data which is sent to the Tx buffer.
o_InterfaceBus_Pi	Output	2 bits	**Output to the Packet indicator buffer** Packet indictor signals. To mark the start of TLP and DLLP.
o_Receive_ACK	Output	1 bit	**Output to the interface Buffer** 1 bit indicating that the row is received from the interface buffer and fully processed and sent to the Tx buffer. If it's equal 1, Tx Buffer can send another row.
o_Data_valid	Output	1 bit	**Output to the Tx buffer** 1 bit for the two bytes.

2.2.3.3. Tx Buffer

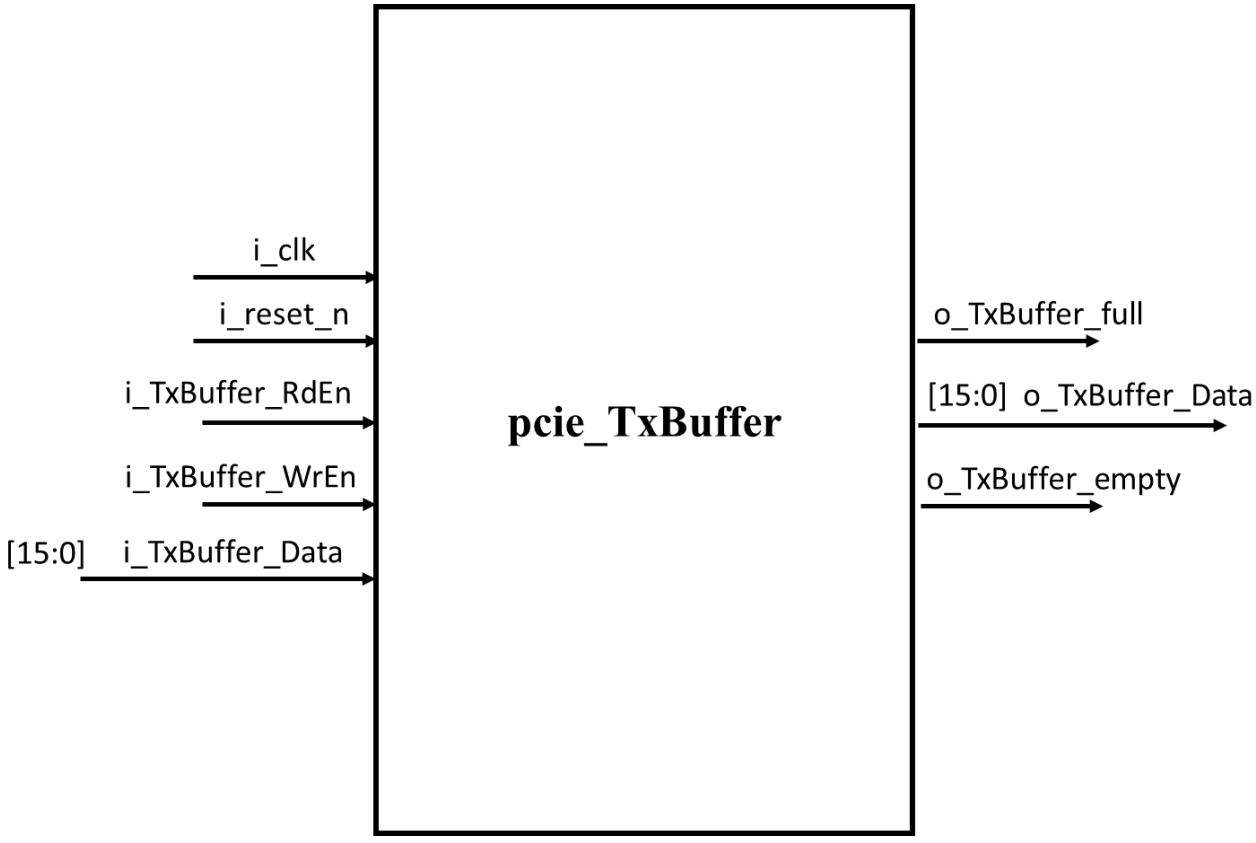

Figure 35: Tx Buffer module

Block main functionality:

The module is a FIFO with 2047 rows depth and two bytes width. It takes the packet two bytes by two bytes from the interface bus and send it to the MUX when it takes the permission from the controller.

The block diagram of the Tx buffer is shown in Figure 35. Table 21 illustrates the signals of the Tx buffer.

Table 21: Tx_Buffer Signals

Name	Direction	Size	Description
i_clk	Input	1 bit	Clock signal for positive edge registers
i_reset_n	Input	1 bit	Global asynchronous reset.
i_TxBuffer_RdEn	Input	1 bit	**Input from the controller** Read enable of the buffer. When it is 1, the buffer outs two bytes to the MUX.
i_TxBuffer_WrEn	Input	1 bit	**Input from the interface bus** Write enable of the buffer. When it is 1, the interface bus inserts two bytes in the buffer.
i_TxBuffer_Data	Input	16 bits	**Input from the interface bus** Two bytes of the packet.
o_TxBuffer_Data	Output	16 bits	**Output to the MUX** Two bytes of the packet.
o_TxBuffer_full	Output	1 bit	**Output to the interface bus** Full flag to indicate that the buffer is full.
o_TxBuffer_empty	Output	1 bit	**Output to the controller** Empty flag to indicate that the buffer is empty.

2.2.3.4. Packet Indicator Buffer

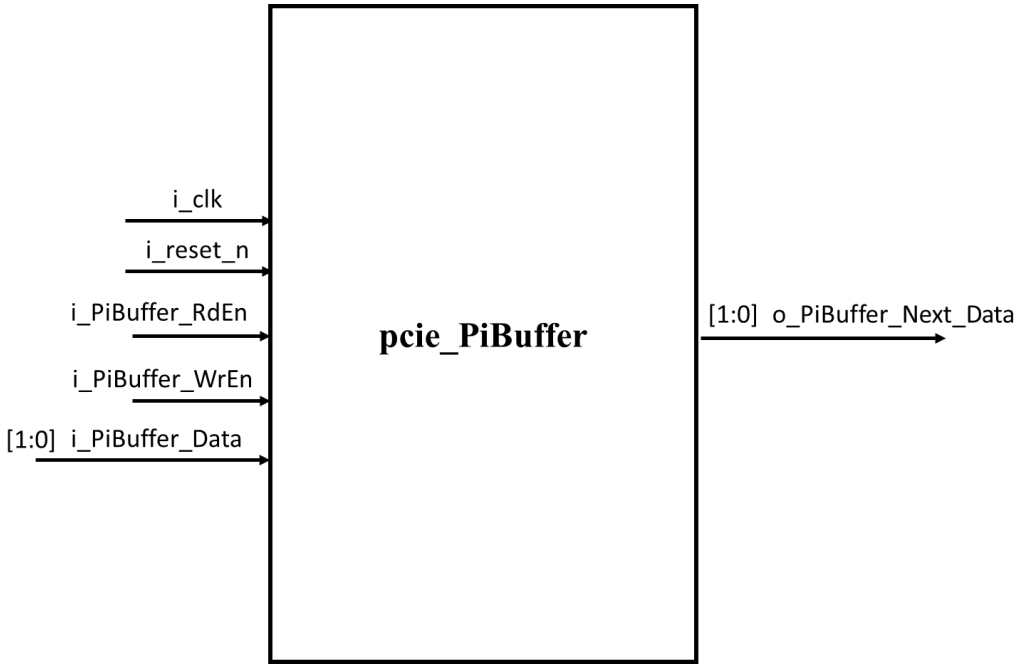

Figure 36: Packet indicator buffer module

Block main functionality:

This module takes the packet indicator signals which indicate the start of packet either it is TLP or DLLP to help Start – End Framing module to capsulate each packet correctly. These packet indicator signals are in parallel with the data in the Tx buffer. Each bit refers to the corresponding byte in the Tx buffer. The block diagram of the packet indicator buffer is shown in Figure 36. Table 22 illustrates the signals of the packet indicator buffer.

Table 22: Packet_Indicator_Buffer Signals

Name	Direction	Size	Description
i_clk	Input	1 bit	Clock signal for positive edge registers
i_reset_n	Input	1 bit	Global asynchronous reset.
i_PiBuffer_RdEn	Input	1 bit	**Input from the controller** Read enable of the buffer. When it is 1, the controller reads the data in the buffer.
i_PiBuffer_WrEn	Input	1 bit	**Input from the interface bus** Write enable of the buffer. When it is 1, the interface bus inserts two bits in the buffer to indicate the start of packet and its type.
i_PiBuffer_Data	Input	2 bits	**Input from the interface bus** Two bits to indicate the start of packet and its type. 00 -> Not valid 01 -> Start of TLP 10 -> Start of DLLP 11 -> Valid data, but not start of packet
o_PiBuffer_Next_Data	Output	2 bits	**Output to the controller** Two bits to indicate the start of packet and its type.

2.2.3.5. Start – End Framing

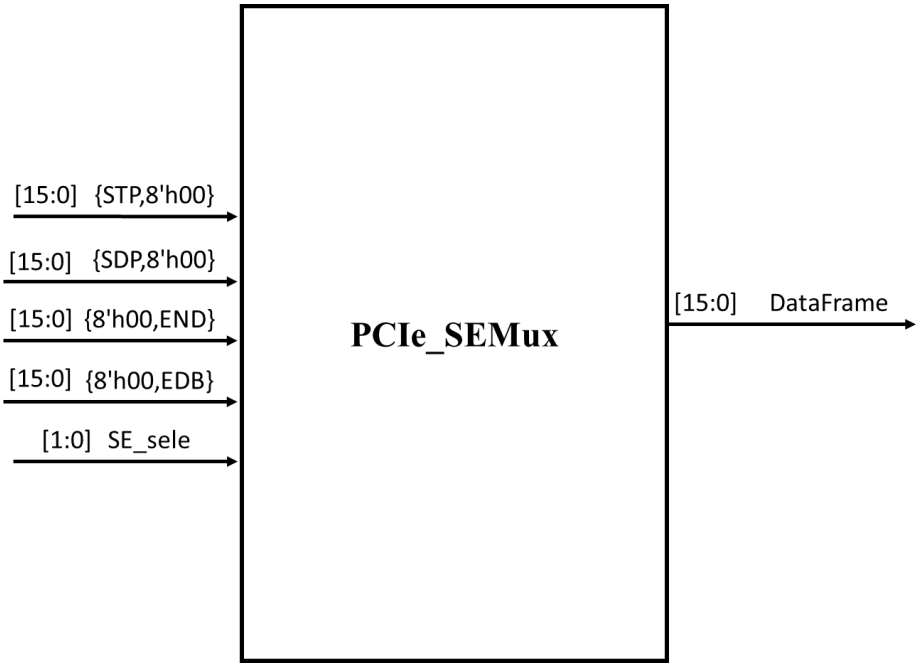

Figure 37: Start End Framing module

Block main functionality:

It is a MUX that select one of 4 different choices that are described in Table 23.

The block diagram of the start – end framing is shown in Figure 37.

Table 23: Start_End_Framing Signals

Name	direction	Size	Description
{STP, 8'h00}	Input	16 bits	Start of TLP
{SDP, 8'h00}	Input	16 bits	Start of DLLP
{8'h00, END}	Input	16 bits	End of the correct TLP or DLLP
{8'h00, EDB}	Input	16 bits	End of the TLP or DLLP with an error. This choice is not used in the design.
SE_sele	Input	2 bits	**Input from the controller** The selection line of this multiplexer. 00 -> STP 01 -> SDP 10 -> END 11 -> EDB
DataFrame	Output	16 bits	**Output to the multiplexer**

Start and end characters are 8 bits but the bus width is 16 bits. So, 8-bit zeros are concatenated with each character. These concatenated bits are overhead and they are not a part of the actual packet. Hence, they will be removed by the framing alignment module.

2.2.3.6. Logical Idle

It the Tx buffer and the Ordered set buffer are empty; the controller should choose to get the data from this module. This module is a fixed 16-bit of zeros with D/K signal equals 0.

2.2.3.7. Ordered set Buffer

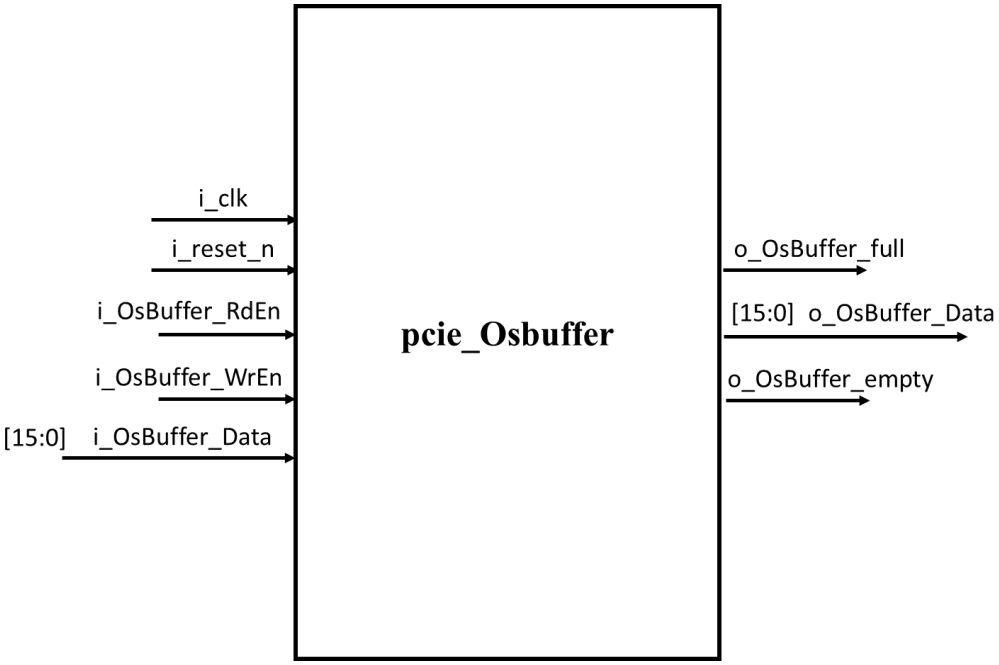

Figure 38: Ordered set Buffer module

Block main functionality:

This is the interface between LTSSM and the Tx block. The LTSSM inserts the ordered set packets like TS1, TS2 and SKP in this buffer whenever it wants.The block diagram of the ordered set buffer is shown in Figure 38. Table 24 illustrates the signals of the ordered set buffer.

Table 24: Ordered_Set_Buffer Signals

Name	Direction	Size	Description
i_clk	Input	1 bit	Clock signal for positive edge registers
i_reset_n	Input	1 bit	Global asynchronous reset.
i_OsBuffer_RdEn	Input	1 bit	**Input from the controller** Read enable of the buffer. When it is 1, the buffer outs two bytes to the MUX.
i_OsBuffer_WrEn	Input	1 bit	**Input from the LTSSM** Write enable of the buffer. When it is 1, the LTSSM inserts two bytes in the buffer.
i_OsBuffer_Data	Input	16 bits	**Input from the LTSSM** Two bytes of the ordered set packet.
o_OsBuffer_Data	Output	16 bits	**Output to the MUX** Two bytes of the ordered set packet.
o_OsBuffer_full	Output	1 bit	**Output to the LTSSM** Full flag to indicate that the buffer is full.

o_OsBuffer_empty	Output	1 bit	**Output to the controller** Empty flag to indicate that the buffer is empty.

2.2.3.8. Controller

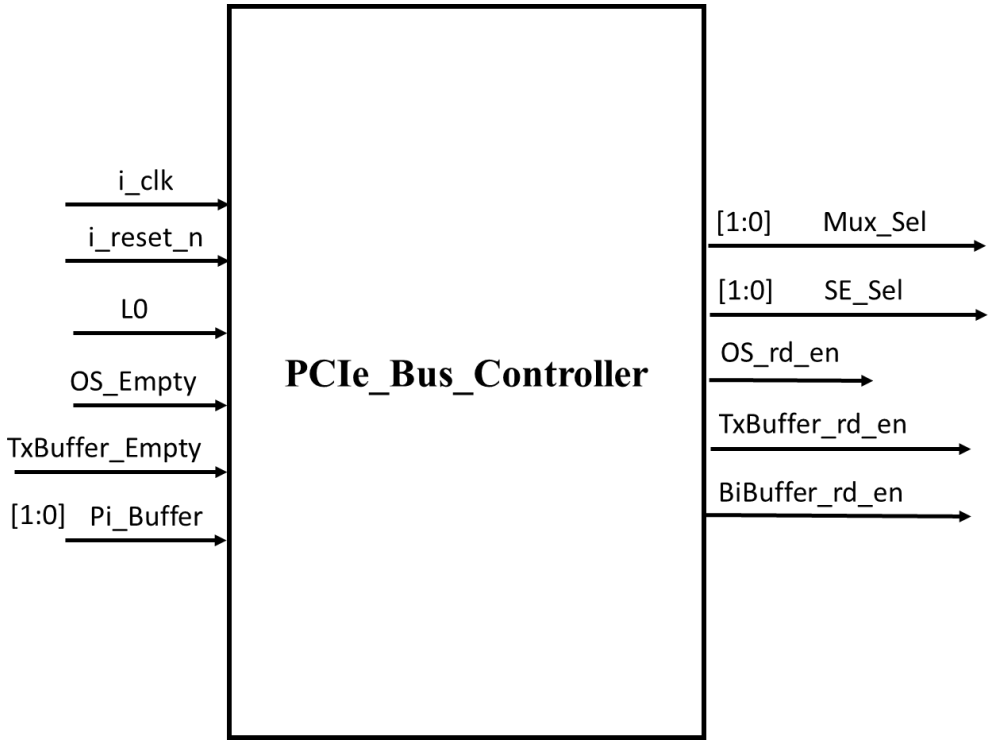

Figure 39: Controller module

Block main functionality:

It is the brain of the Tx block as it controls the all operations of the Tx block. It receives status signals from the Tx buffer, Packet Indicator buffer, Ordered set buffer and LTSSM. It sends control signals to the MUX, Tx buffer, Packet Indicator buffer, Ordered Set buffer and the Start – End framing.

The function of the controller can be described in the following points:

- If L0 equals zero, the MUX can't take data from Tx buffer.
- If there is a data on the ordered set buffer, the MUX should take this data until the ordered set buffer gets empty.
- If the ordered set buffer is empty and the Tx buffer is empty, the MUX should select the logical idle.
- If the ordered set buffer is empty and the Tx buffer is not empty, the MUX should select the Start – End framing with the correct choice then take the data from the Tx buffer.
- If a new data comes to the ordered set buffer while the MUX selects the Tx buffer, the MUX should still select the Tx buffer till the current packet is finished with END then select the ordered set buffer.

The block diagram of the controller is shown in Figure 39.

Table 25 illustrates the signals of the controller.

Table 25: Controller Signals

Name	Direction	Size	Description
i_clk	Input	1 bit	Clock signal for positive edge registers
i_reset_n	Input	1 bit	Global asynchronous reset.
L0	Input	1 bit	**Input from the LTSSM** If it equals 1, the normal operation mode is activated. If it equals 0, LTSSM is still in link training or initialization mode.
OS_Empty	Input	1 bit	**Input from the Ordered set buffer** If it equals 1, the ordered set buffer is empty. If it equals 0, the ordered set buffer is not empty.
TxBuffer_Empty	Input	1 bit	**Input from the Tx buffer** If it equals 1, the Tx buffer is empty. If it equals 0, the Tx buffer is not empty.
Pi_Buffer	Input	2 bits	**Input from the packet indicator buffer** The two bits data of the packet indicator.
Mux_Sel	Output	2 bits	**Output to the Multiplexer** The selection line of the multiplexer.
SE_Sel	Output	2 bits	**Output to the Start – End Framing** The selection line of the multiplexer of the start – End Framing.
OS_rd_en	Output	1 bit	**Output to the Ordered set buffer** If it equals 1, the ordered set buffer outs two bytes of its data.
TxBuffer_rd_en	Output	1 bit	**Output to the Tx buffer** If it equals 1, the Tx buffer outs two bytes of its data.
BiBuffer_rd_en	Output	1 bit	**Output to the Packet indicator buffer** If it equals 1, the Packet indicator buffer outs two bytes of its data.

2.2.3.9. Multiplexer

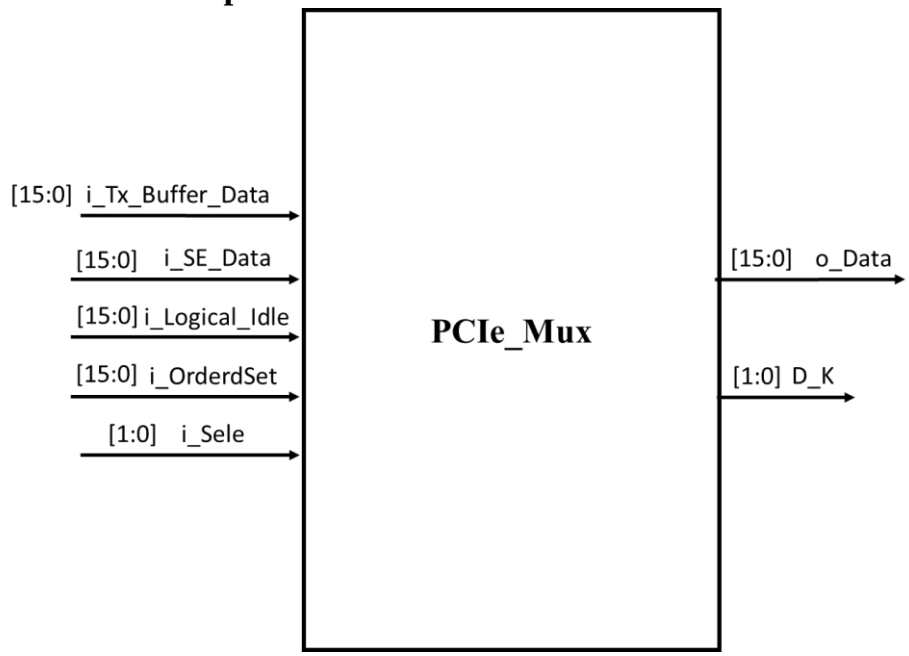

Figure 40: Multiplexer module

Block main functionality:

This multiplexer should choose to take the two bytes data from four different modules. These modules are the Tx buffer, the start – end framing, the logical idle and the ordered set buffer. There are two outputs of this module. The first one is two bytes data while the second one is two bits representing the type of each output bytes whether it is data or control character.

The block diagram of the multiplexer is shown in Figure 40. Table 26 illustrates the signals of the multiplexer.

Table 26: Multiplexer Signals

Name	Direction	Size	Description
i_Tx_Buffer_Data	Input	16 bits	**Input from the Tx buffer** The two bytes data of the packet.
i_SE_Data	Input	16 bits	**Input from the Start – End Framing** The two bytes of the Start – End characters.
i_Logical_Idle	Input	16 bits	**Input from the logical idle** The two bytes of the logical Idle characters.
i_OrderdSet	Input	16 bits	**Input from the ordered set buffer** The two bytes of the ordered set packets.
i_Sele	Input	2 bits	**Input from the controller** The selection line of the multiplexer.
o_Data	Output	16 bits	**Output to the Framing Alignment** Two Bytes data.

D_K	Output	2 bits	**Output to the Framing Alignment**
			One bit for each byte.
			If (1): it means the corresponding byte is control.
			If (0): it means the corresponding byte is data.

2.2.3.10. Framing Alignment

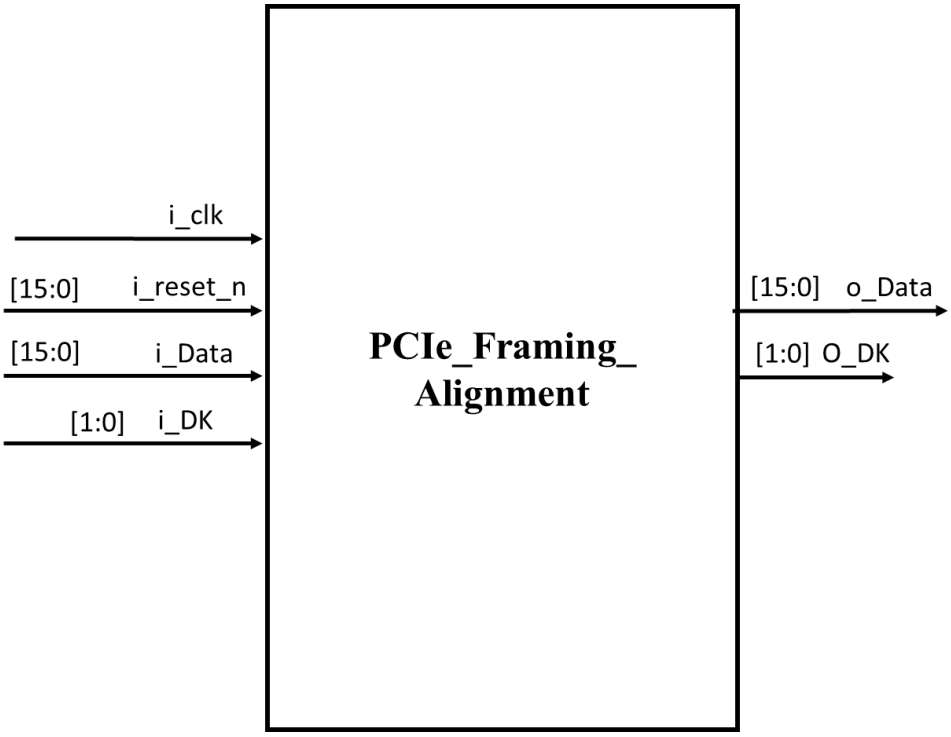

Figure 41: Framing Alignment module

Block main functionality:

The function of this module is to remove the overhead zeros that are added in start – end framing module. The block diagram of the multiplexer is shown in Figure 41.

Table 27 illustrates the signals of the framing alignment.

Table 27: Framing_Alignment Signals

Name	Direction	Size	Description
i_clk	Input	1 bit	Clock signal for positive edge registers
i_reset_n	Input	1 bit	Global asynchronous reset.
i_Data	Input	16 bits	**Input from the multiplexer** Two Bytes data.
i_DK	Input	2 bits	**Input from the multiplexer** One bit for each byte. If the bit equals 0, it means the corresponding byte is data. If the bit equals 1, it means the corresponding byte is control.
o_Data	Output	16 bits	**Output to the PIPE** Two Bytes data.
o_DK	Output	2 bits	**Output to the PIPE** One bit for each byte. If the bit equals 0, it means the corresponding byte is data. If the bit equals 1, it means the corresponding byte is control.

2.3. Rx

2.3.1. Rx overview

The main objective for the receiver (RX) block in the physical layer is to reconstruct the incoming bytes from the pipe interface to one complete packet and pass it to the upper layers. If any order set is received, it should not propagate to the upper layers and only sent to LTSSM to handle it as shown in Figure 42: Receiver block. Also, the Rx block logical part is responsible for handling different data types besides ignoring any garbage or wrong data packets incoming from the electrical part through the pipe interface.

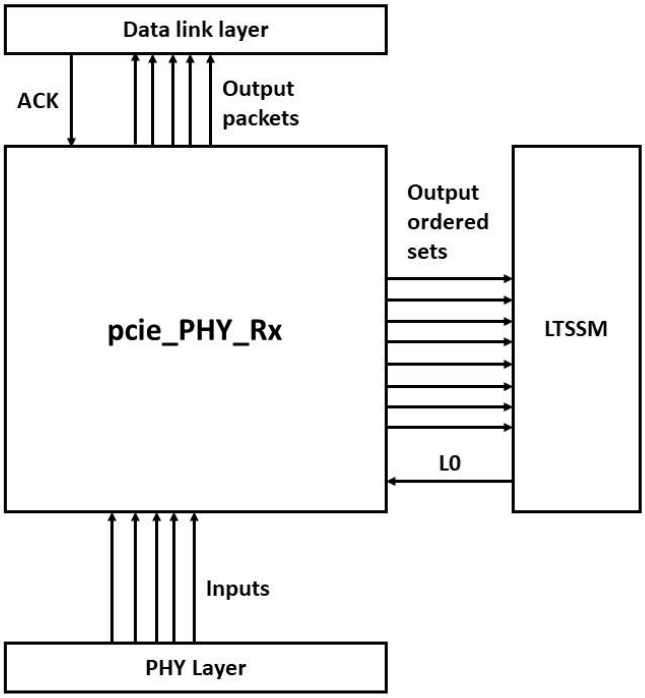

Figure 42: Receiver block diagram

2.3.2. Rx interfaces with PIPE, LTSSM and Data Link Layer

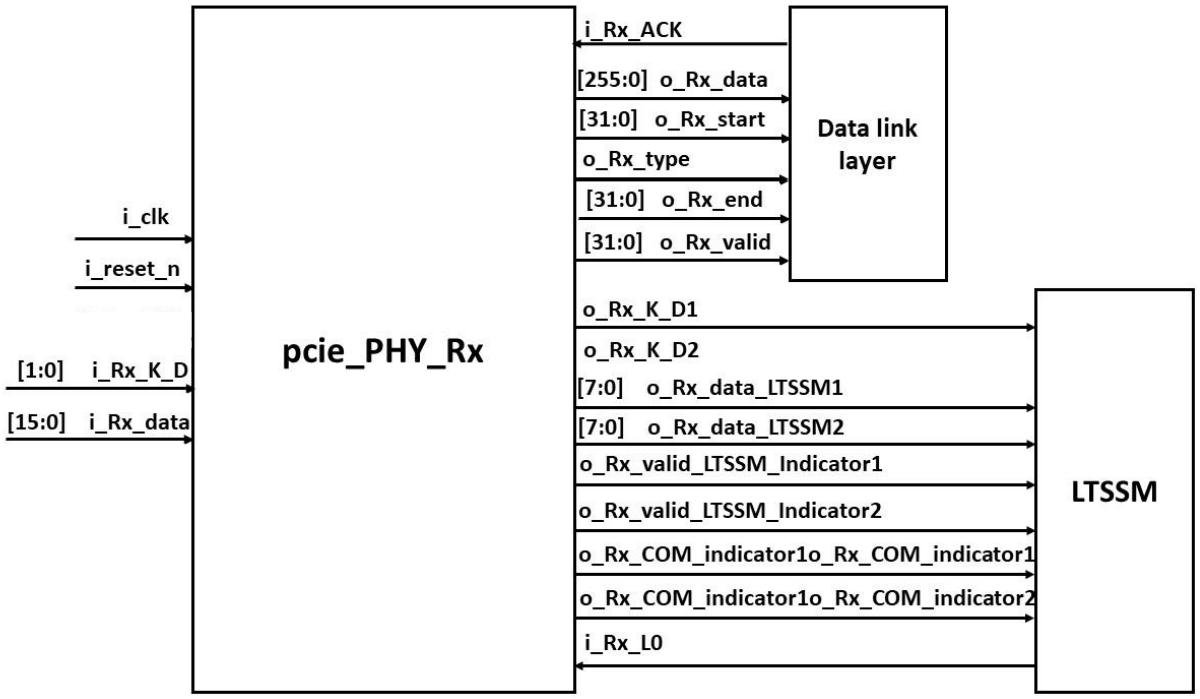

Figure 43: Receiver interfaces

The top module is connecting all input and output signals of all blocks together to integrate their functions and behave as one block, it also interfaces with the LTSSM block, the data link layer, and the phi layer. Figure 43 shows the interface signals with the three main blocks.

2.3.2.1. PIPE Interface

Every clock cycle two bytes of decoded data is received from the PHY layer electrical part by the pipe interface with two i_Rx_K_D signals. Full details for each input and output signal is described in Table 28.

Table 28: Rx-PIPE interface signals

Name	Direction	Size	Description
i_clk	Input	1bit	**Input from PIPE Interface.** Clock signal for positive edge registers.
i_reset_n	Input	1 bit	**Input from PIPE Interface.** Global asynchronous reset.

i_Rx_K_D	Input	2 bits	**Input from PIPE Interface.**
			K_D signal indicates whether the input character from PHY layer is data or control character; the least significant bit is an indicator for the first byte of data and the second bit is an indicator for the second byte of data. For each bit:
			If (1): the input character is data character.
			If (0): input character is control character

i_Rx_data	Input	16 bits	**Input from PIPE Interface.** 16 bits (2 bytes) Input data from PHY layer

2.3.2.2. Data Link Layer Interface

When a start of TLP or DLLP is filtered, the next data character is stored temporarily in a buffer until an end of the packet is filtered. After receiving a complete packet, it is sent to the data link layer in multiple frames on consecutive clock cycles. Each frame consists of 32 bytes in size, and it is sent to a data link with start, end, valid and type indicators. Each consists of 32 bits; each bit describes the state of the corresponding byte. Full details for each input and output signal is described in Table 29.

Table 29: Rx-Data Link Layer interface signals

Name	Direction	Size	Description

i_Rx_ACK	Input	1 bit	**Input from Data Link Layer Interface.**
			If (1): the data link layer has received the packet and it can receive the next packet.
			If (0): the data link layer can't receive another packet

o_Rx_data	Output	256 bits	**Output to Data Link Layer Interface.** 32-byte output data.
o_Rx_start	Output	32 bits	**Output to Data Link Layer Interface.** 32 mapping bits, if any bit = (1), that indicates the start of a packet. If o_Rx_start [31] = 1: Indicates that a start of the packet at byte 31 of the packet.
o_Rx_end	Output	32 bits	**Output to Data Link Layer Interface.** 32 mapping bits, if one bit of its bits is equal to 1, this indicates the end of the packet at the corresponding byte position in the 32 bytes of the output data.
o_Rx_valid	Output	32 bits	**Output to Data Link Layer Interface.** 32 mapping bits, if any bit of its bits is equal to 1, this indicates that the corresponding byte position in the 32 bytes of the output data is a valid byte and is a part of the packet.
o_Rx_type	Output	1 bit	**Output to Data Link Layer Interface.** 1-bit indicates the type of the received packet. • If (0): this a TLP packet. • If (1): this a DLLP packet.

2.3.2.3. LTSSM Interface

When a com character is filtered an o_Rx_COM_indicator signal is activated and sent to the LTSSM block that indicates a start of an ordered set. All the next characters are sent to the LTSSM block with an o_Rx_valid_LTSSM_Indicator to indicates it is a valid byte that goes to the LTSSM until a start of TLP or DLLP is filtered, then the o_Rx_valid_LTSSM_Indicator is deactivated and the sending process is stopped. Each byte is sent to LTSSM has a valid signal (o_Rx_valid_LTSSM_Indicator), COM indicator (o_Rx_COM_indicator) and a K/D signal (o_Rx_K_D), to make it easier to decode those characters in LTSSM. Full details for each input and output signal is described previously in LTSSM interface with Transmitter, Receiver block and Pipe chapter and in Table 7: LTSSM-Rx interface signals.

2.3.3. Rx hardware description

The Rx block consists mainly of the Rx filter, the Rx buffer as shown in Figure 44. This design is done to achieve the main functionality of the Rx Block in the PHY layer logical part:

1. Filter all control characters to not reach to the upper layers.
2. Forwarding the incoming ordersets to LTSSM.
3. Perform reliable packet transfer at the interface with physical layer and data link layer by defining ACK signal comes from data link layer.
4. Eliminate the error packets that doesn't start with STP or SDP control characters or doesn't end with END or EBD control characters.
5. Response under the LTSSM control if the link is not in normal operation yet, or the device enter a recovery state.

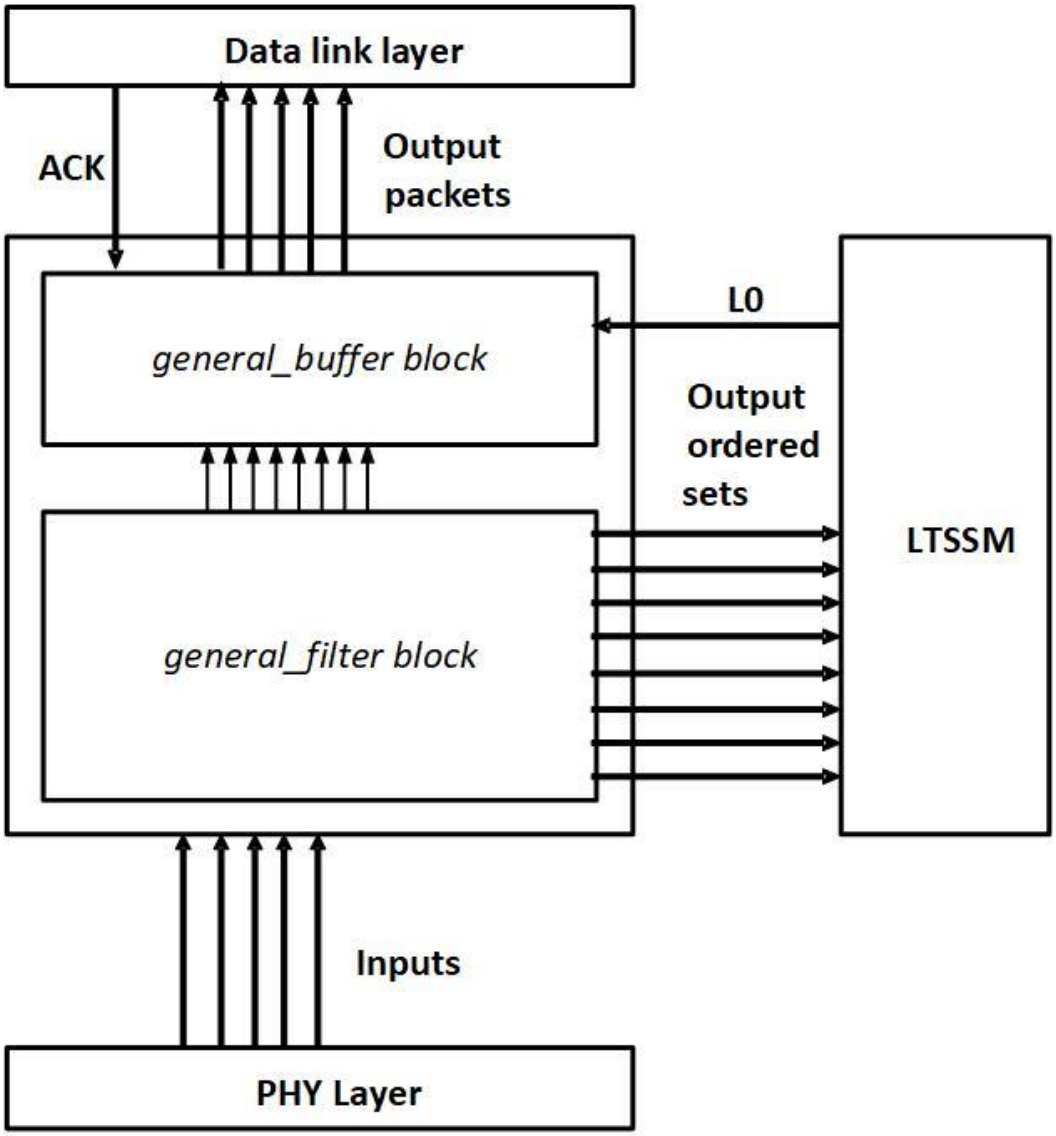

Figure 44: Receiver Block implementation

2.3.3.1. General filter Block

It is the block which decides whether the incoming characters from the PHY layer have to be sent to the LTSSM or data link layer. The incoming serial byte stream coming from the electrical part of the physical layer contains TLPs, DLLPs, Logical Ideal sequences and control characters such as STP, SDP, END, PADs as well as the ordered sets. Of these, the Logical Ideal sequence, the control characters and the ordered sets are detected and sent to the LTSSM and not allowed to be sent to the DLL. The TLPs and the DLLPs are filtered and sent to the upper layer. The previous functionality is done by the RX Filter as shown in Figure 45.

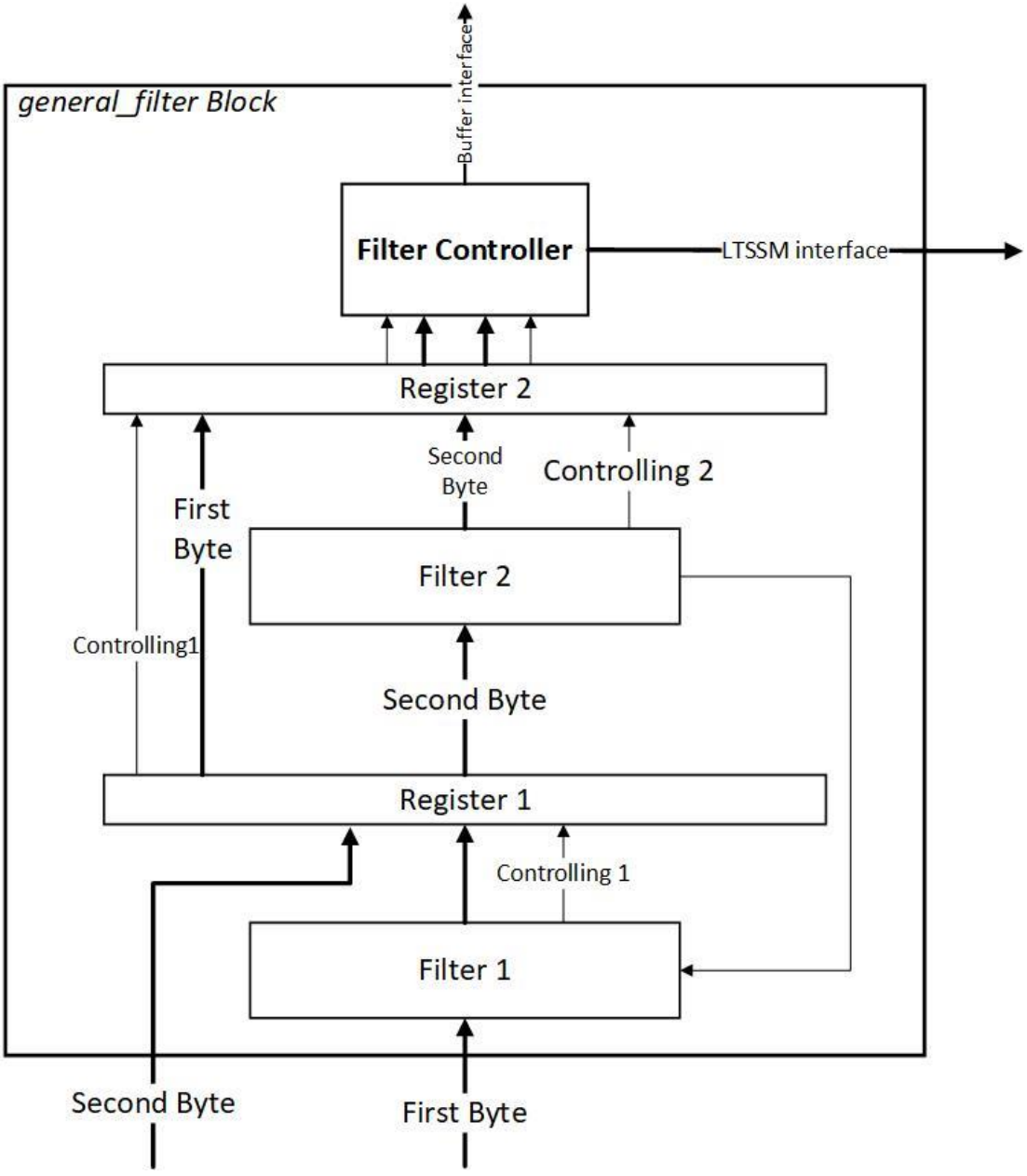

Figure 45: general_filter implementation

Table 30 shows the input output signals of the general filter block.

Table 30: general_filter signals

Name	Direction	Size	Description
i_clk	Input	1 bit	**Input from PIPE Interface.** Clock signal for positive edge registers.
i_reset_n	Input	1 bit	**Input from PIPE Interface.** Global asynchronous reset.

i_generalFilter_K_D1	Input	1 bit	**Input from PIPE Interface.**
			K_D signal indicates whether the first input character from the PHY layer is data or control character.
			If (1): the input character is data character.
			If (0): input character is control character

i_generalFilter_data1	Input	8 bits	**Input from PIPE Interface.** 8 bits (1 bytes) The least input byte coming from PHY layer

i_generalFilter_K_D2	Input	1 bit	**Input from PIPE Interface.**
			K_D signal indicates whether the second input character from the PHY layer is data or control character.
			If (1): the input character is data character.
			If (0): input character is control character

i_generalFilter_data2	Input	8 bits	**Input from PIPE Interface.** 8 bits (1 bytes) The most input byte coming from PHY layer
o_generalFilter_Data_Enable1	Output	1 bit	**Input from general_filter block.** Enable signal indicates if the first byte has to be forwarded to LTSSM or to Data Link Layer. ● If (0): this symbol is forwarded to LTSSM, so it will not be handled by the buffer modules. ● If (1): this symbol is forwarded to DLL.
o_generalFilter_Data_Enable2	Output	1 bit	**Input from general_filter block.** Enable signal indicates if the second byte has to be forwarded to LTSSM or to Data Link Layer. ● If (0): this symbol is forwarded to LTSSM, so it will not be handled by the buffer modules. ● If (1): this symbol is forwarded to DLL.
o_generalFilter_data_Rx_buffer1	Output	8 bits	**Input from general_filter block.** The first byte coming from general_filter.
o_generalFilter_data_Rx_buffer2	Output	8 bits	**Input from general_filter block.** The second byte coming from general_filter.
o_generalFilter_controlsSignals1	Output	3 bits	**Input from general_filter block.** The control signal that defines the state of the symbol 1. We classify each symbol to one of the following states:

Value	Indicates to
000	Error Byte (ERR)
001	Start of TLP (STP)
010	Start of DLLP (SDP)
011	END character (END)
100	END bad character (EBD)
101	Valid Byte (VLD)
111	Default state or not valid (DFT)

<table>
<tr><td>o_generalFilter_
controlsSignals2</td><td>Output</td><td>3 bits</td><td>Input from general_filter block.
The control signal that defines the state of the symbol 2.
We classify each symbol to one of the following states:
<table><tr><th>Value</th><th>Indicates to</th></tr><tr><td>000</td><td>Error Byte (ERR)</td></tr><tr><td>001</td><td>Start of TLP (STP)</td></tr><tr><td>010</td><td>Start of DLLP (SDP)</td></tr><tr><td>011</td><td>END character (END)</td></tr><tr><td>100</td><td>END bad character (EBD)</td></tr><tr><td>101</td><td>Valid Byte (VLD)</td></tr><tr><td>111</td><td>Default state or not valid (DFT)</td></tr></table></td></tr>
<tr><td>o_genertalFilter_
data_LTSSM1</td><td>Output</td><td>8 bits</td><td>Output to LTSSM
The least byte of data.</td></tr>
<tr><td>o_genertalFilter_
data_LTSSM2</td><td>Output</td><td>8 bits</td><td>Output to LTSSM
The most byte of data.</td></tr>
<tr><td>o_genertalFilter_
valid_LTSSM_I
ndicator1</td><td>Output</td><td>1 bit</td><td>Output to LTSSM
it is a valid signal for the least byte of data.
● if 0 : the least byte is not valid.
● if 1 : the least byte is valid.</td></tr>
<tr><td>o_genertalFilter_
valid_LTSSM_I
ndicator2</td><td>Output</td><td>1 bit</td><td>Output to LTSSM
it is a valid signal for the most byte of data.
● if 0 : the most byte is not valid.
● if 1 : the most byte is valid.</td></tr>
<tr><td>o_genertalFilter_
COM_indicator1</td><td>Output</td><td>1 bit</td><td>Output to LTSSM
Indicator signal to indicate if the first byte is the beginning of order set.
● if 0 : it is not a COM character.
● if 1 : it is a COM character.</td></tr>
<tr><td>o_genertalFilter_
COM_indicator2</td><td>Output</td><td>1 bit</td><td>Output to LTSSM
Indicator signal to indicate if the first byte is the beginning of order set.
● if 0 : it is not a COM character.
● if 1 : it is a COM character.</td></tr>
<tr><td>o_genertalFilter_
K_D1</td><td>Output</td><td>1 bit</td><td>Output to LTSSM
K_D signal indicates whether the first byte goes to LTSSM is data or control character.
● if 0 : the first byte is a control character.
● if 1 : the first byte is a data character.</td></tr>
<tr><td>o_genertalFilter_
K_D2</td><td>Output</td><td>1 bit</td><td>Output to LTSSM
K_D signal indicates whether the first byte goes to LTSSM is data or control character.
● if 0 : the first byte is a control character.
● if 1 : the first byte is a data character.</td></tr>
</table>

2.3.3.2. Filter Block

The Rx filter operates as a finite state machine (FSM) to filter the incoming byte as shown in Figure 46, it goes from one state to another until it completes the full functionality of the Rx filter.

Due to that Gen1 link handles 2 incoming bytes each clock cycle so, two blocks of the filter block is used to handle them by handling one byte by each block, a feedback signal contains the next state of filter2 is feeded to the current state of filter1 and it is initially equivalent to state0.

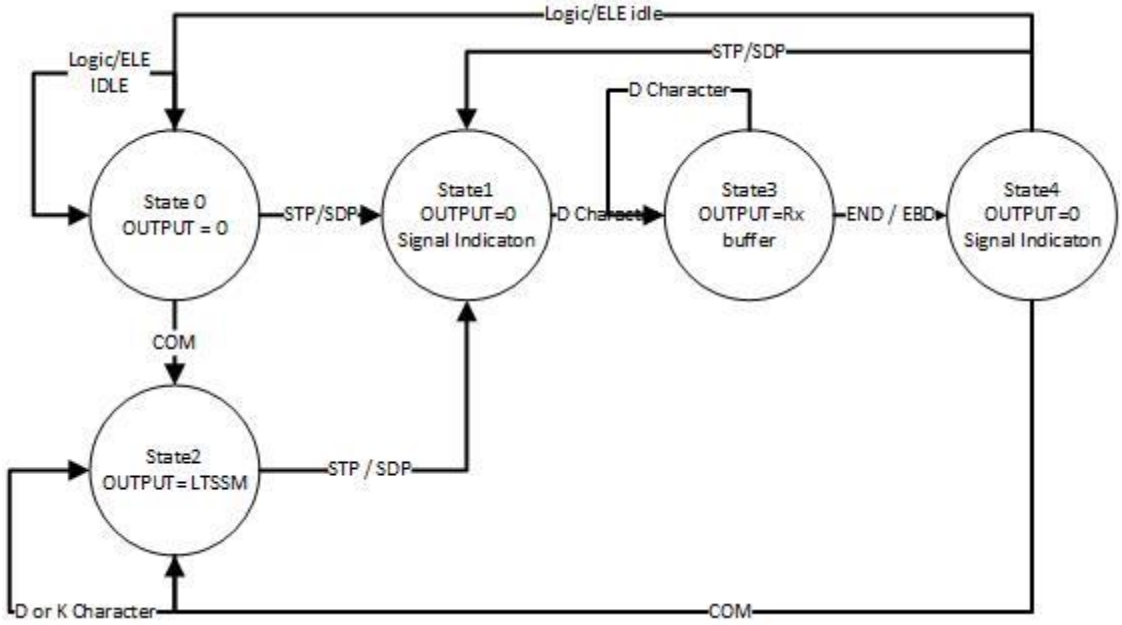

Figure 46: The FSM of the Rx filter logic

2.3.3.3. Register Block

Two registers are used to keep the synchronization between the two filters and handle the 2 bytes each clock cycle.

2.3.3.4. Filter controller

The filter controller rule is to take the last stage decision in the block and decide whether the 2 bytes will be branched to the LTSSM or go up the Rx Buffer, and this is done using the output controlling signals from Filter1 and Filter2.

2.3.3.5. General Buffer Block

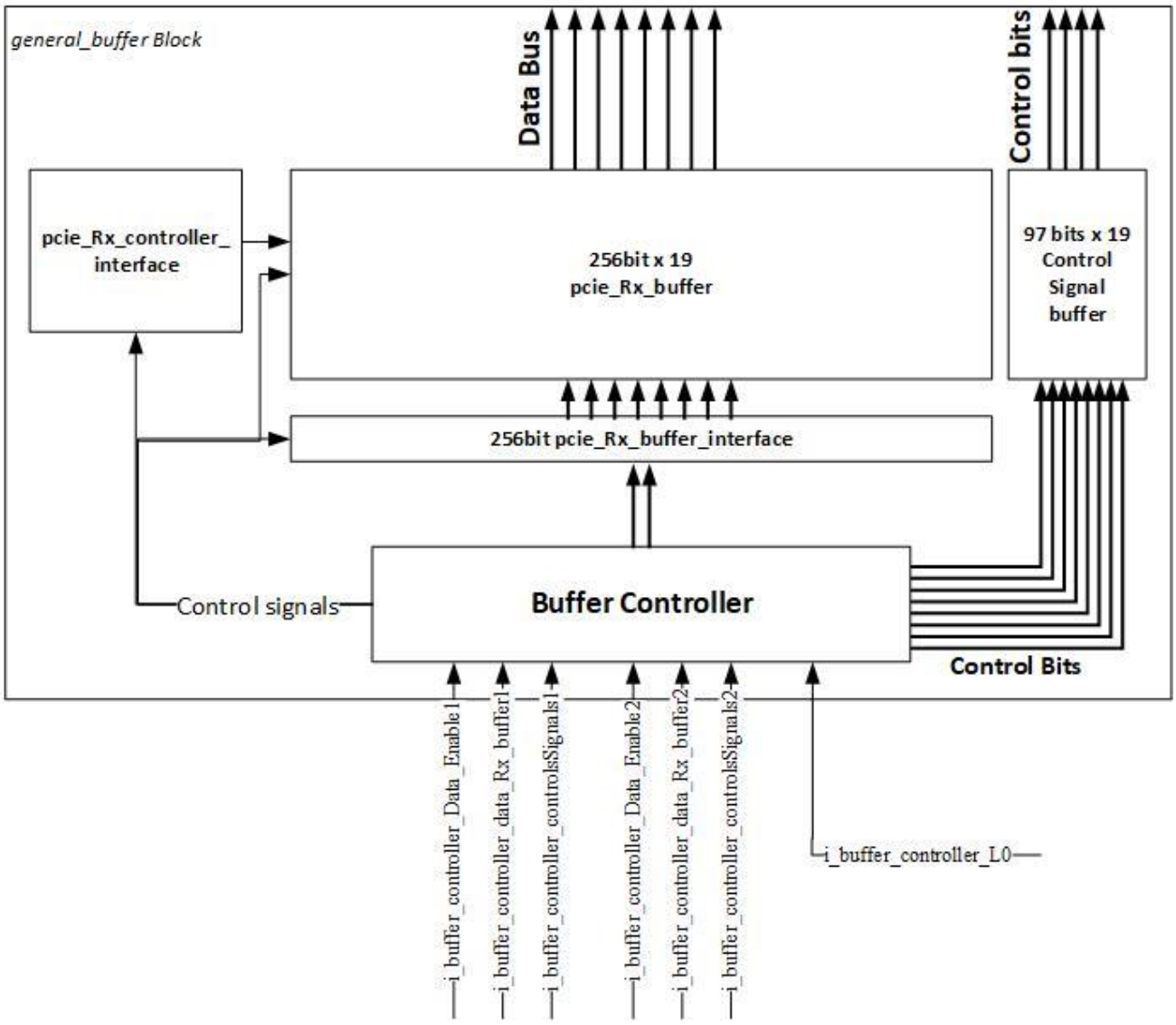

Figure 47: general_buffer implementation

2.3.3.6. Buffer Controller

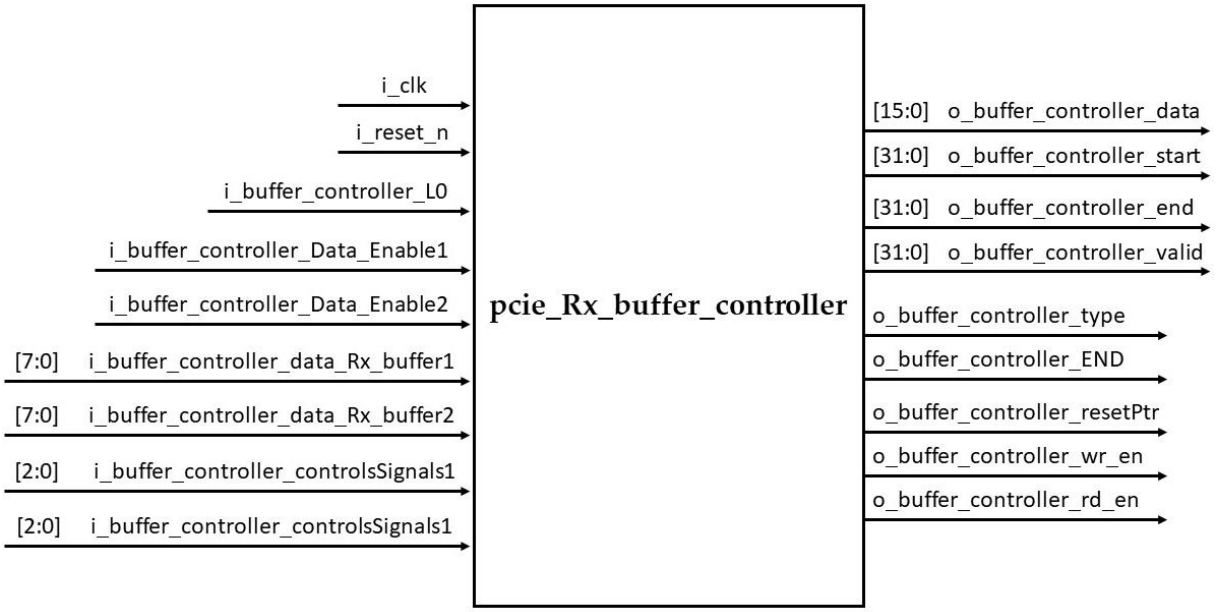

Figure 48: buffer controller module

Block main functionality:

As shown in Figure 48, the buffer controller takes 2 bytes from the filter block and their control signal:

- Data_Enable signal: that indicates if the corresponding byte is valid to propagate to the upper layers or not.
- ControlSignals signal: that indicates the state of the corresponding byte, we had 7 states that described in the filter section. (ERR, STP, SDP, END, EBD, VLD, DFT)

The buffer controller module is responsible to check on Data_Enable first, if '1' so this byte had to be handled and somehow must propagate to Data link layer. It checks second on the state of this byte and update the internal registers that store these states.

The controller also counts the number of the received valid bytes, when it receives 32 bytes or receive END symbol, whatever what is coming first, it sends the start, end, valid, type states to be stored in the control signal buffer. The 32 bytes of data is stored in the buffer_interface module.

When END symbol is received, the controller enables the output o_buffer_controller_END that tells the controller_interface module a complete packet is received.

Input and outputs:

Table 31 shows inputs and outputs for the buffer controller module.

Table 31: Buffer_Controller signals

Name	Direction	Size	Description
i_clk	Input	1bit	Clock signal for positive edge registers.
i_reset_n	Input	1 bit	Global asynchronous reset.
i_buffer_controller_L0	Input	1 bit	**Input from LTSSM block.** L0 signal indicates if the LTSSM reach successfully to L0 state or not yet. • If (1): the link in the normal operation state L0 • If (0): the link in link training states, no packet is received.
i_buffer_controller_Data_Enable1	Input	1 bit	**Input from general_filter block.** Enable signal indicates if the first byte has to be forwarded to LTSSM or to Data Link Layer. • If (0): this symbol is forwarded to LTSSM, so it will not be handled by the buffer modules. • If (1): this symbol is forwarded to DLL.
i_buffer_controller_Data_Enable2	Input	1 bit	**Input from general_filter block.** Enable signal indicates if the first byte has to be forwarded to LTSSM or to Data Link Layer. • If (0): this symbol is forwarded to LTSSM, so it will not be handled by the buffer modules. • If (1): this symbol is forwarded to DLL.
i_buffer_controller_data_Rx_buffer1	Input	7 bits	**Input from general_filter block.** The first byte coming from general_filter.
i_buffer_controller_data_Rx_buffer2	Input	7 bits	**Input from general_filter block.** The second byte coming from general_filter.
i_buffer_controller_controlsSignals1	Input	2 bits	**Input from general_filter block.** The control signal that defines the state of the symbol 1.
i_buffer_controller_controlsSignals2	Input	2 bits	**Input from general_filter block.** The control signal that defines the state of the symbol 2.
o_buffer_controller_data	Output	16 bits	**Output to buffer_interface module.** 2-bytes of data which passes to buffer.
o_buffer_controller_start	Output	32 bits	**Output to control_signal_buffer module.** 32 mapping bits, if any bit = (1), that indicates the start of a packet.
o_buffer_controller_end	Output	32 bits	**Output to control_signal_buffer module.** 32 mapping bits, if any = (1), that indicates End of packet.
o_buffer_controller_valid	Output	32 bits	**Output to control_signal_buffer module.** 32 mapping bits, for all bits that = (1) are indicate those are valid bytes, normally these valid bytes must be between start bit and end bit.
o_buffer_controller_type	Output	1 bit	**Output to control_signal_buffer module.** 1-bit indicates the type of the received packet. • If (0): this a TLP packet. • If (1): this a DLLP packet.
o_buffer_controller_END	Output	1 bit	**Output to controller_interface module.**

			It indicates that the controller received END symbol, so the controller_interface knows there is a complete packet is received.
o_buffer_controller_resetPtr	Output	1 bit	**Output to buffer_interface module.** If the controller received END symbol, it resets the pointer in the buffer interface.
o_buffer_controller_wr_en	Output	1 bit	**Output to buffer_interface module.** It enables the next module to write in the fifo or not.
o_buffer_controller_rd_en	Output	1 bit	**Output to buffer_interface and buffer modules.** It enables the stored data in the buffer interface to be read and storing in the next buffer. Normally it becomes ‘1’ when 32 bytes are received, or END symbol is received.

2.3.3.7. Buffer Interface

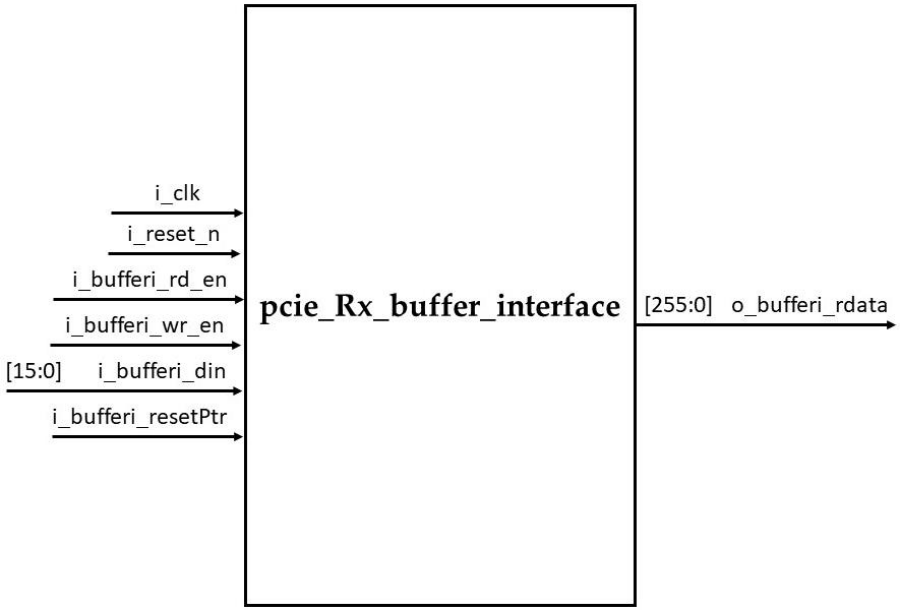

Figure 49: buffer interface module

Block main functionality:

The size of the buffer interface is 32 bytes. Each clock cycle, it takes 2 bytes from the buffer controller and *wr_en* signal, which is '1' when controller knows these are two valid bytes, and store them till this buffer full. When receiving 32 bytes the *rd_en* signal high to forward all these 32 bytes to the next buffer. To reset the pointer position when END symbol is received in *buffer_controller*, a *resetPtr* signal is exist.

Input and outputs:

Table 32 shows inputs and outputs for the buffer interface module.

Table 32: Buffer_interface signals

Name	Direction	Size	Description
i_clk	Input	1bit	Clock signal for positive edge registers.
i_reset_n	Input	1 bit	Global asynchronous reset.
i_bufferi_rd_en	Input	1 bit	**Input from buffer_controller module.** It is control signal from the controller that enables the interface buffer to read from it.
i_bufferi_wr_en	Input	1 bit	**Input from buffer_controller module.** Write enable signal that makes the buffer to store the data.
i_bufferi_din	Input	16 bits	**Input from buffer_controller module.** The input data that should be stored in the buffer.
i_bufferi_resetPtr	Input	1 bit	**Input from general_filter block.** It is a control signal that reset the fifo position, mainly if the controller received END byte.
o_bufferi_rdata	Output	256 bits	**Output to buffer module.** 32 bytes are passed to the buffer to be stored.

2.3.3.8. Buffer

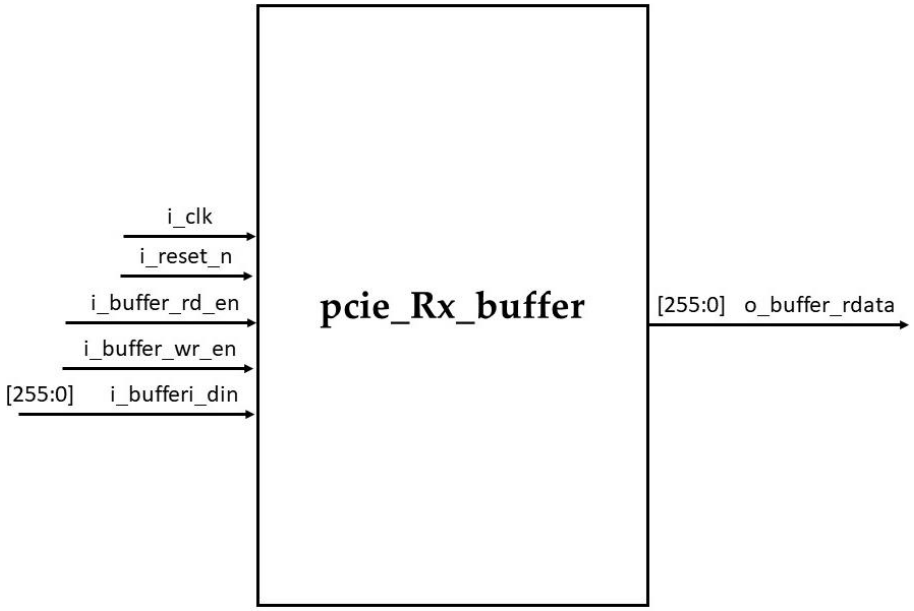

Figure 50: buffer module

Block main functionality.

It is the main buffer that receive 32 bytes until the complete packet is received, then there is another controller that enable the buffer to out its stored data in sequence of 32 bytes also to the data link layer.

The buffer is 32 bytes width and 19-line depth. These parameters as the maximum supported packet is 544 bytes.

Input and outputs.

Table 33 shows inputs and outputs for the buffer module.

Table 33: buffer signals

Name	Direction	Size	Description
i_clk	Input	1bit	Clock signal for positive edge registers.
i_reset_n	Input	1 bit	Global asynchronous reset.
i_buffer_rd_en	Input	1 bit	**Input from controller_interface module.** It is control signal from the controller interface that enables the buffer to read from it.
i_buffer_wr_en	Input	1 bit	**Input from buffer_controller module.** Write enable signal that makes the buffer to store the data in its input port.
i_buffer_din	Input	256 bits	**Input from buffer_interface module.** The input data that should be stored in the buffer.
o_buffer_rdata	Output	256 bits	**Output to Data Link Layer.** 32 bytes are passed to the DLL.

2.3.3.9. Control Signal Buffer

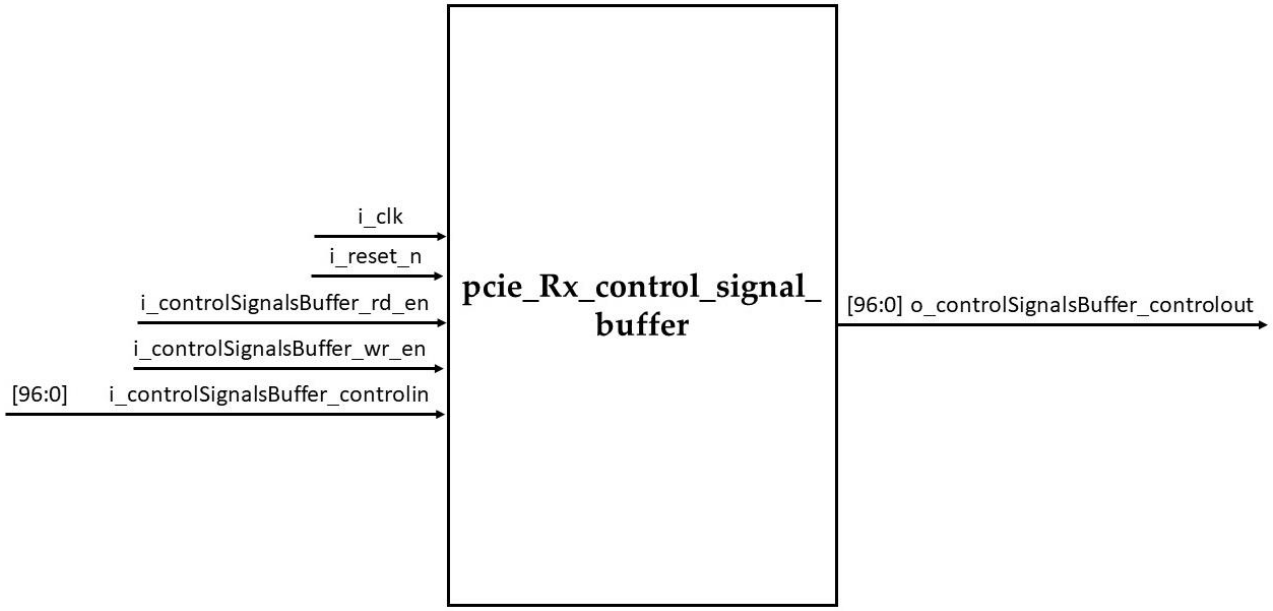

Figure 51: Control Signal Buffer module

Block main functionality:

It is the buffer that stores the state of the transfer bytes which goes to data link layer. This module is very similar to the buffer module, its width is 97 bits as (32-bits start, 32-bits end, 32-bits valid, 1-bit type) and its depth 19 also. The read and write pointer is moving as read and write pointers in the buffer module.

Input and outputs:

Table 34 shows inputs and outputs for the control signal buffer module.

Table 34: Control_Signal_Buffer signals

Name	Direction	Size	Description
i_clk	Input	1bit	Clock signal for positive edge registers.
i_reset_n	Input	1 bit	Global asynchronous reset.
i_controlSignalsBuffer_rd_en	Input	1 bit	**Input from controller_interface module.** It is control signal from the controller interface that enables the buffer to read from it.
i_controlSignalsBuffer_wr_en	Input	1 bit	**Input from buffer_controller module.** Write enable signal that makes the buffer to store the data in its input port.
i_controlSignalsBuffer_controlin	Input	97 bits	**Input from buffer_controller module.** The input data that should be stored in the buffer.
o_controlSignalsBuffer_controlout	Output	97 bits	**Output to Data Link Layer.** 97 indicators bits are passed to the DLL.

2.3.3.10. Controller Interface

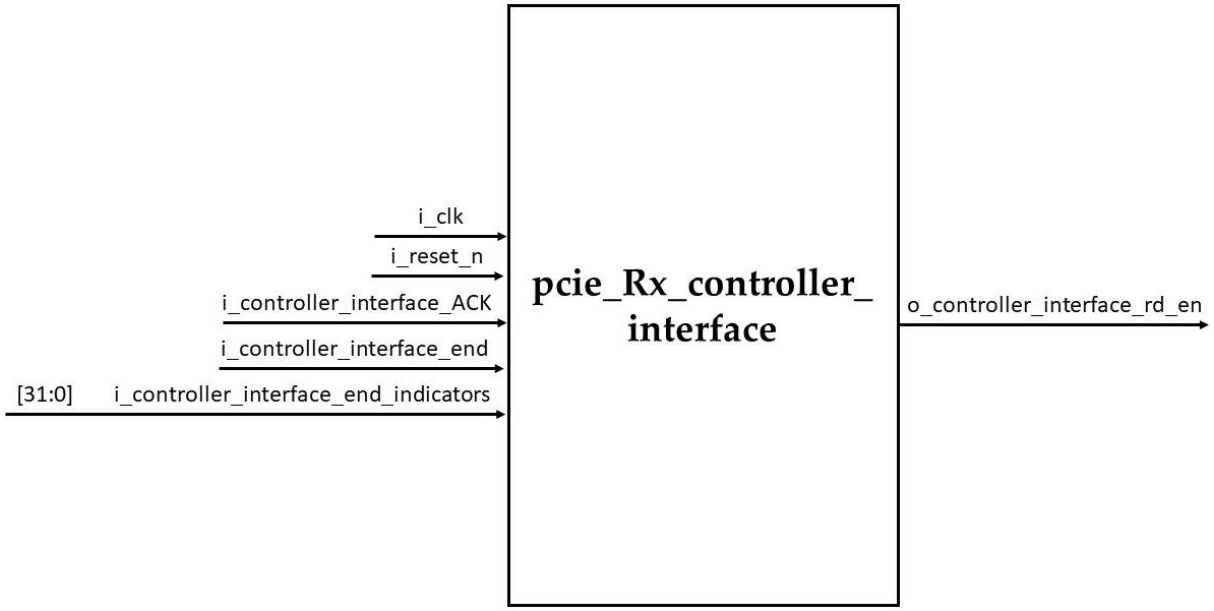

Figure 52: Controller Interface module

Block main functionality:

This module is responsible to control data transfer between buffers and Data Link Layer. It runs under a state finite machine that shown in the next figure.

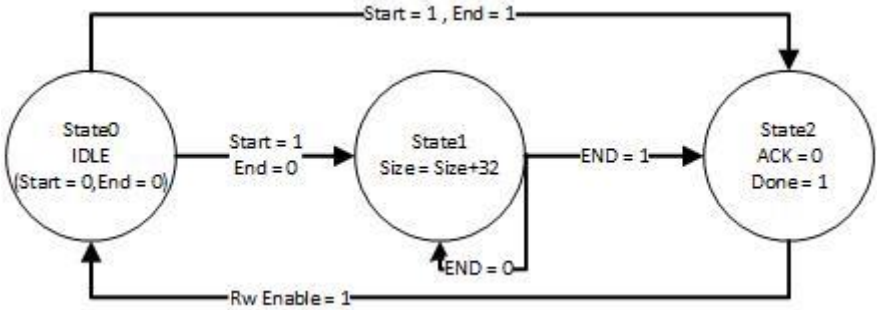

Figure 53: The FSM of the Controller interface logic

- There are two internal counters. Countre1: is increased if there is new complete packet is received, in another word, if i_controller_interface_end ='1' the counter is increased. Counter2: is increased if the transfer of the packet is completed.
- State0 IDLE state: as well as the counter1 = counter2, so the number of the received packets is equal to the number of the transferred packets, so the controller still exists in this state until counter1 is increased and become counter1 not equal to counter2.
- If counter1 ≠ counter2 → the controller goes to state1 or state2 according to the number of clock cycle that transfer needs.
- If the packet size ≤ 32 bytes, so it could transfer in one clock cycle. The next state is state2. If the packet size > 32 bytes, so it needs more than one clock cycle. The next state is state1.
- In state1: the controller exists in this state enabling the rd_en in the buffer and control signal buffer modules until the transfer of the packet is completed.

- In state2: this state performs as a HOLD state, the controller is holds there at least one clock cycle to check on the ACK signal coming from data link layer interface, if ACK =’1’ the controller back to state0. If Ack = ‘0’ the controller waits in this state until receiving ACK.

Input and outputs:

Table 35 shows inputs and outputs for the controller interface module.

Table 35: Controller_Interface signals

Name	Direction	Size	Description
i_clk	Input	1bit	Clock signal for positive edge registers.
i_reset_n	Input	1 bit	Global asynchronous reset.
i_controller_interface_ACK	Input	1 bit	**Input from Data Link Layer.** It indicates if the data link layer can receive other packets or not.
i_controller_interface_end	Input	1 bit	**Input from buffer_controller module.** It indicates that the buffer stored a complete packet.
i_controller_interface_end_indicators	Input	32 bits	**Input from controlSignalBuffer module.** 32 bits that indicates if there a END of packet in these 32 bytes or not.
o_controller_interface_rd_en	Output	1 bit	**output to buffer and controlSignalBuffer module.** Read enable signal that control the buffer and controllersignalbuffer modules to out their stored data on the bus.

3. Testing

3.1. Block level testing

3.1.1. LTSSM testing

In this section, the testing procedure used to test the LTSSM functionality is discussed.

3.1.1.1. Time scaling

As discussed in the implementation of the stateMachine module part, there are different numbers of OS required to be sent and received to fulfill the exit condition from a state to another state, along with different timeout values. These OS required numbers and timeout values are stated in the PCIe spec.

For the simulation process to be realistic, these values need to be scaled down to ensure that simulation does not take the whole day to finish. This procedure does not affect the functionality test but it is done to make the testing and debugging easier.

Table 36: Timeouts and OS scaling shows the scaled timeout values, the required number of the OS to be sent or received in each substate along with the values from the spec.

Table 36: Timeouts and OS scaling

State	Timeout		Sent Ordered sets		Received Ordered sets	
	Spec (ms)	Scaled (clk cycles)	Spec	scaled	Spec	Scaled
Detect.Quiet	12	12	--	--	--	--
Detect.Active	12	12	--	--	--	--
Polling.Active	24	260	1024	24	8	8
Polling.Config	48	160	16	16		
Config.Linkwidth.start	24	24	--		2	2
Config.Linkwidth.Accept	24	24	--		2	2
Config.Lanenum.wait	2	20	--		2	2
Config.Lanenum.Accept	2	20	--		2	2
Config.Complete	2	160	16	16	8	8
Config.Idle	2	160	16 (Idle)	16	8 (Idle)	8

Note that in Config.Idle substate, the values in the sent and received OS table represent the number of sent and received logical idle symbols, not OS.

3.1.1.2. LTSSM states numbering

Table 37 shows the numbers assigned to each substate. This helps to recognize the states in the when they appear in the waveform.

Table 37: States numbering

States	Number (hex)
Detect.Quiet	0
Detect.Active	1
Polling.Active	2
Polling.Config	3
Config.Linkwidth.start	4
Config.Linkwidth.Accept	5
Config.Lanenum.wait	6
Config.Lanenum.Accept	7
Config.Complete	8
Config.Idle	9
L0	A

3.1.1.3. Test plan

After completing the design of each module contained in the LTSSM design (OS_Creator, Timer, OS_Decoder, …), a test have been carried out on each module to test its functionality which is fully described in the implementation section. After that, Integration of the tested block took place to give us the LTSSM block.

Verification was done on two steps to test the functionality of the full integrated LTSSM block:

1- Forced testing, which means the injection of Training sequences TS1s and TS2s in different patterns to check the transitions of the states and compare it to the PCIe spec transitions.
2- Back-to-Back testing, which means connecting two LTSSM blocks together in a way that they negotiate the Training sequences automatically to reach the full operation state L0.

Figure 54 shows a simple diagram that represents the way to have a full functional LTSSM block after passing the integration and the testing processes.

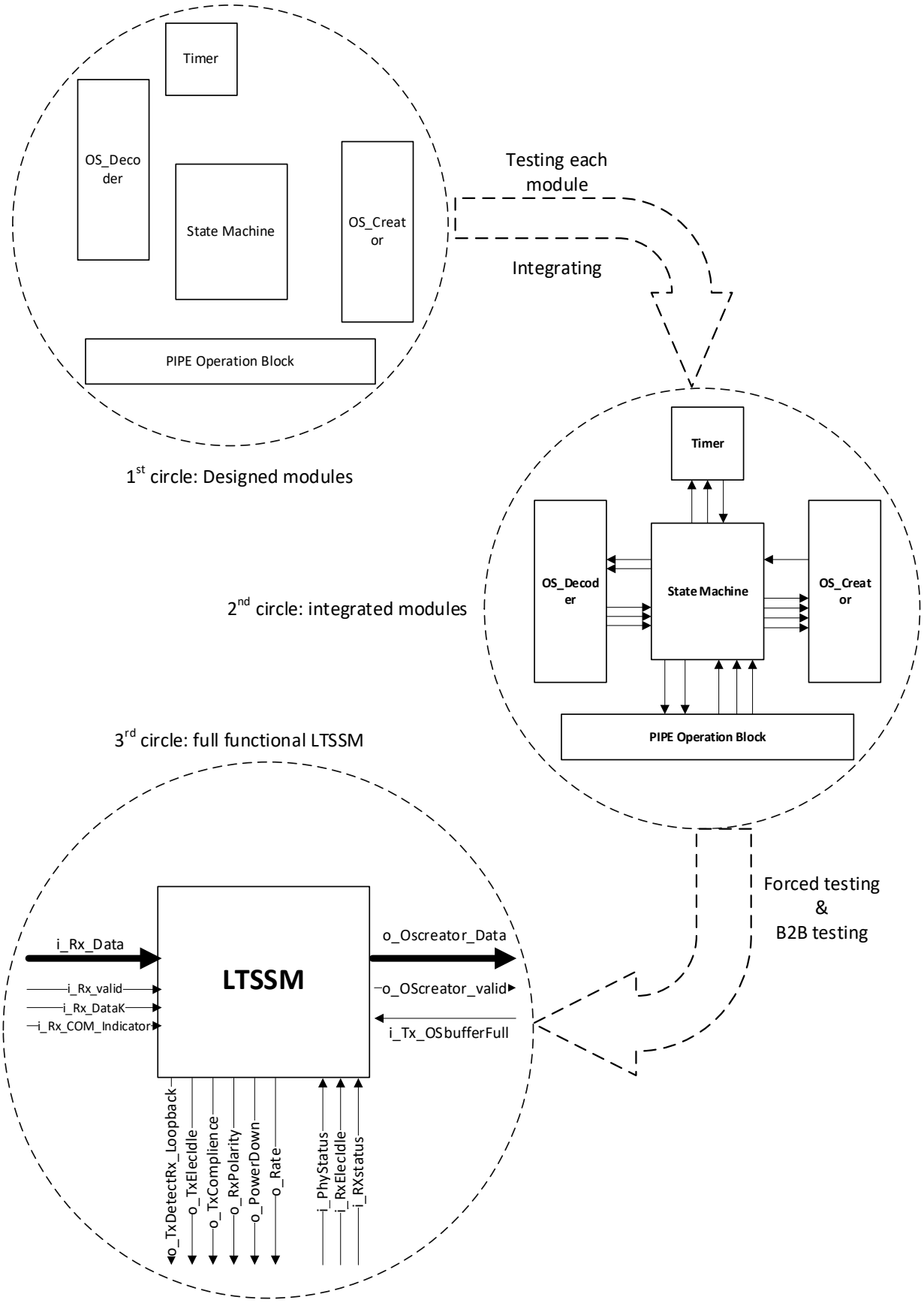

Figure 54: LTSSM design path

1- Forced testing

In this part, we perform positive and negative testing. For positive testing, we have three expected behaviors from the LTSSM, which are:

- **Normal operation case**: the transition from state to another to reach the full operation state L0.

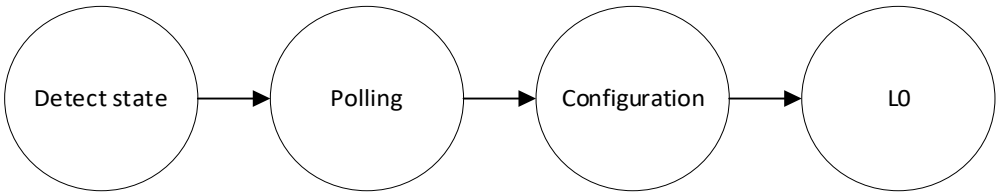

Figure 55: normal operation path

- **Timeout case:** when time out occurs in any state, we need to reach the detect state and start again the normal operation from detect state.

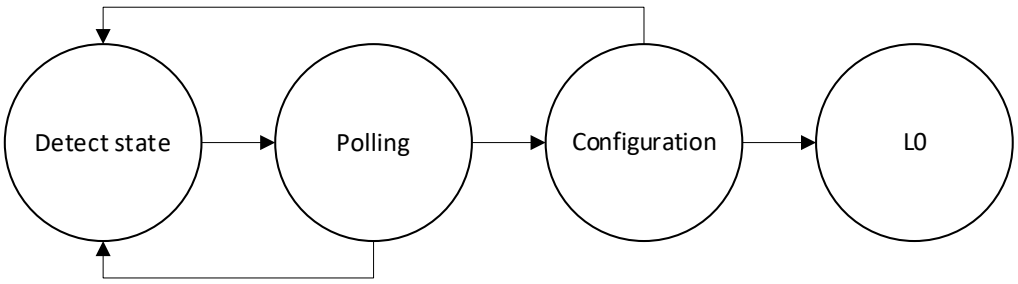

Figure 56: timeout path

- **Buffer of Tx is full case:** In any state if i_Tx_OSbufferFull=1, so we stop transmitting the order sets until the TX buffer is not full. When the i_Tx_OSbufferFull =0, we start to complete the order set from the point we stopped at.

Note: according to our design, in training sequence the Transmitter must be fully dedicated to link training, so the buffer can't be full.

Table 38 shows the positive testcases performed to test the LTSSM functionality. Where Table 39 shows the negative testcases.

The negative testcases show cases that would never happen in the normal operation of the block, but it is done to ensure that the block would not be stuck if any error happed during its operation.

Table 38: LTSSM positive test cases

Test feature	Test scenario	Expected output	Output
Normal operation	• **Detect state:** lane detected • **Polling active:** receive 8 TS1 with train control 0 & send >=1024 TS1 • **From Polling configuration to configuration idle:** normal TS negotiation	Normal operation Case (1)	Pass
Normal operation	• **Detect state:** lane detected • **Polling active:** receive 8 TS1 with train control 4 & send >=1024 TS1 • **From Polling configuration to configuration idle:** normal TS negotiation	Normal operation Case (1)	Pass
Normal operation	• **Detect state:** lane detected • **Polling active:** receive 8 TS2 & send >=1024 TS1 • **From Polling configuration to configuration idle:** normal TS negotiation	Normal operation Case (1)	Pass
Full Tx buffer	• **Detect state:** lane detected • FULL Tx=1 • FULL Tx=0	Full Tx case Case (3)	Pass
Time out case	• Link not detected	Time out case Case (2)	Pass
Time out case	• **Detect state:** lane detected • **Polling active:** time out without satisfying the exit condition.	Time out case Case (2)	Pass
Time out case	• **From detect to configuration idle:** normal detection and TS negotiation. • **Configuration idle:** time out without satisfying the exit condition	Time out case Case (2)	Pass

Table 39: LTSSM negative test cases

Test feature	Test scenario	Output
Change link number negotiated during the train sequence from state to another	• Detect state: lane detected • From polling active to configuration linkwidth start: normal negotiation • Configuration linkwidth accept: here we change the link number negotiated before	Pass
Change link number negotiated during the train sequence in the same state.	• Detect state: lane detected • From polling active to polling configuration: normal negotiation • Configuration link width start: we receive 2 TS1 with different link width number.	Pass

Back-to-Back testing

In this part, two blocks are connected together, one as an upstream component and one as a downstream, as shown in Figure 57.

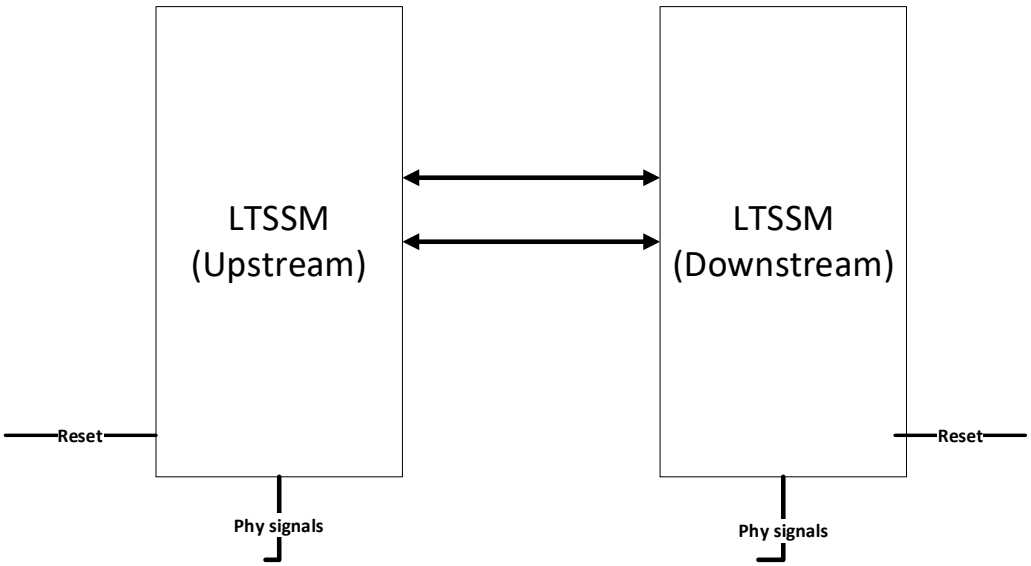

Figure 57: LTSSM Back-to-Back integration

The state transition in this test is fully automatic, except for the transition from Detect state to Polling state. This is done by emulating the PIPE signal to both of the LTSSM blocks, starting form Polling state, we can start observing the state transitions.

Back-to-Back testing criteria

- We start by reset the 2 LTSSM
- Emulating the PIPE detect sequence.
- Start observing the state transitions.

State transitions observation

1. first the condition required for lane detection satisfy (Phystatus=1, Rxstatus=3), so lane detected.
2. In polling states, TS1 negotiation starts.

3. In configuration states, the downstream starts to negotiate link and lane width number and wait for the upstream response.
4. In L0 state, we start to send SKIP order sets each 1180 symbols time.

Note:The exit condition of Upstream from Config.Lanenum.wait to Config.Lanenum.Accept is to Received 2 TS2 with the link saved and Lane saved. So upstream will be delayed by 2 states to maintain this condition as the downstream will send TS2 with link and lane save at Conifg.complete.

3.1.2. Tx testing

3.1.2.1. Introduction

Testing *Tx block* is based on loading a memory with an input file data which contains test vectors for the features that is being tested, then writing the outputs to an output file and comparing it with some golden output as shown in Figure 58. The main features of Tx module have been tested in L0 state and also during link training and initialization step to ensure that the module behavior is consistent with the standard.

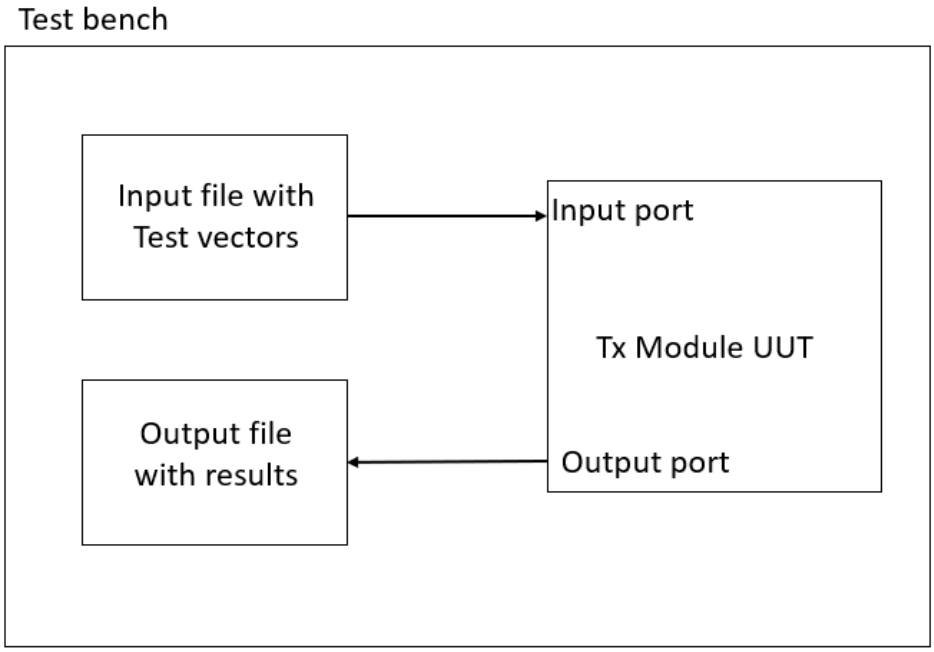

Figure 58: Tx module Test environment

List of features to be tested as expected input in both link training and normal state (positive test)

1. Handling sequence of TLPs and DLLPs of different sizes.
2. Handling reset in different scenarios.
3. Sending logical idle if no data from LTSSM nor the Data link layer.
4. Handling OS packets priority on the TLPs/DLLPs.
5. Handling the transition between link training and L0 state (normal state).

List of features to be tested as not expected (negative test)

1. Data link layer sends TLPs or DLLPs while link training (L0=0).
2. Data link layer sends a packet with valid discontinuity.
3. Data link layer sends a packet with 2 consecutive starts of packet.
4. Data link layer sends a packet without EOP.
5. Data link layer sends a packet without SOP.

Table 40 shows the positive testcases performed to test the Tx functionality and Table 41 show a real scenario to be run, while Table 42 shows the negative testcases.

3.1.2.2. Test plan elements

Table 40: Tx positive test cases

	Test Feature	Test Scenario	Expected Output	Pass/Fail	Notes
1	Reset	Pushing reset at any time	Output symbols are 0000 and DK=11.	PASSED	
2	L0=1, Sending TLP	Min. sized TLP after reset (OS empty)	TLP is sent with ‘FB’ and ‘FD’ as framing (STP,END)characters. All DK=11, except the symbol contains STP=’FB’ DK=10 and the symbol contains END=’FD’ DK=01.	PASSED	
3	L0=1, Sending TLP	Arbitrary sized TLP after reset (OS empty)	Same as feature 2	PASSED	
4	L0=1, Sending TLP	Max. sized TLP after reset (OS empty)	Same as feature 2	PASSED	
5	L0=1, Sending DLLP	Sending DLLP after reset (OS empty)	DLLP is sent with ‘5C’ and ‘FD’ as framing (SDP,END)characters. All DK=11, except the symbol contains SDP=’5C’ DK=10 and the symbol contains END=’FD’ DK=01.	PASSED.	
6	L0=1, OS priority	Sending TLP/DLLP and OS in the same time	ALL OS packets will be sent first and then TLP/DLLP are sent	PASSED.	
7	L0=1, OS priority	LTSSM sends data while sending TLP/DLLP	MAC will finish the current TLP/DLLP only then transform to send OS packets	PASSED.	
8	L0=1, OS priority	Data link layer sends TLPs and DLLPs to MAC while sending OS packets	MAC will continue to send OS packets to PHY until they are finished then MAC will start to send TLPs and DLLPs if exists.	PASSED.	
9	L0=1, sending logical idle	No data coming from the data link layer nor the LTSSM	Logical idle will be sent until any data comes to MAC	PASSED.	
10	L0=1, logical idle to data	LTSSM/Data link layer send data while sending logical idle	MAC will send LTSSM/Data link layer Data instead of logical idle	PASSED.	
11	L0=0, sending logical idle	No data from the LTSSM	Logical idle will be sent	PASSED.	

12	L0=0, Sending OS	LTSSM sends data to MAC	OS packets will be sent	PASSED.	
13	L0=0, logical idle to OS	LTSSM sends data to MAC while sending logical idle	Same as feature 11	PASSED.	
14	L0=0, to L0=1	Link training finished while sending OS packets	Same as feature 8	PASSED.	
15	L0=0 to L0=1	Link training finished while sending logical idle and there are no DLLPs/TLPs	MAC will continue to send logical idle	PASSED.	
16	L0=0 to L0=1	Link training finished while sending Logical idle and there are DLLPs/TLPs	MAC will start to send TLPs/DLLPs	PASSED.	
17	L0=1 to L0=0	L0 became 0 while sending OS packets	MAC will continue to send OS packets	PASSED.	
18	L0=1 to L0=0	L0 became 0 while sending logical idle	MAC will continue to send Logical idle until LTSSM sends data	PASSED.	
19	L0=1 to L0=0	L0 became 0 while sending TLP/DLLP	MAC will continue to send the current TLP/DLLP ONLY and then starts to send OS packets if exist and if not, logical idle will be sent	PASSED	
20	Sending data after reset	Reset followed by OS or DLLPs/TLPs	OS or DLLPs/TLPs will be sent	PASSED.	
21	Packet handling	Sending consecutive TLPs and DLPs	TLPs and DLLPs are sent in order as input with STP/SDP and END framing characters	PASSED.	

Table 41: Tx Real Scenario

Test scenario	PASS/FAIL
Reset	ALL PASSED
L0=1, ts1	
TLP	
DLLP	
TLP &Ts2	
DLLP	
TLP	
TLP	
Reset	
L0=0, TLP (negative testing)	
DLLP (negative testing)	
Ts1	
Ts2	
L0=1	
TLP	
DLLP	
DLLP	
Ts1	
L0=0(empty clock cycle)	
L0=1(empty clock cycle)	

3.1.2.3. Negative testing

Table 42: Tx negative test cases

Test Case	**Output behaviour**	**Notes**
Data coming from the Data link layer during link training and initialization.	Data will be stored until the training is finished and then will be sent on the lane	
Data link layer sends Max sized Packet (544 Bytes) without EOP	The buffer will be flushed, packet will be ignored and Data link layer will not receive ACK on the packet	
Data link layer sends Packet without SOP in the first row	Packet will be ignored and Data link layer will not receive ACK on the packet	

Data link layer sends packet with valid discontinuity	Packet will be transmitted until the byte that have valid=0.	
Data link layer sends packet with SOP=32'h C000_0000 instead SOP=32'h 8000_0000	Packet will be transmitted correctly.	
Data link layer sends packet with SOP=32'h E000_0000 instead SOP=32'h 8000_0000	Packet will be transmitted from the last start sent and so on.	
Data link layer sends arbitrary sized packet without EOP.	Packet will not be sent until it receives a row that have EOP only without SOP.	This test case affects the next packet from the data link layer which means the next packet after the described one will not be transmitted correctly.

3.1.3. Rx testing

3.1.3.1. Introduction

In this test plan, the main features and functionality have been tested. The main functionality of the Rx module in the normal operation state (L0 state) is receiving sequences of TLPs and DLLPs with different sizes, filtering and removing the framing symbols then forwarding them to the data link layer. In addition, handling the ordered sets which are coming to LTSSM in Link training and initialization Mode. Most of the expected scenarios has been tested to make sure that the Rx block will operate in real scenarios correctly.

List of features to be tested as an expected input in normal state (positive testing):

1. Handling a sequence of minimum sized packets.
2. Handling a sequence of very large packets.
3. Handling TLPs with all available sizes.
4. Handling a sequence of DLLPs after TLPs with maximum size.
5. Handling Packets that come without resetting the module.
6. Reset the module in different scenarios.
7. Handling all different order sets.
8. Handling packets while activating and deactivating the ACK from the DLL.
9. Handling receiving from PHY layer different types of packets and order sets while disabling L0 signal.

List of features to be tested which are not expected as an input in normal state (negative testing):

1. Handling uncompleted packets (packets without end).
2. Handling packets without start.
3. Handling packets with size of odd number of bytes.
4. Handling packets with STP/SDP and END/EBD, but started from second byte.
5. Handling incoming OS in different fault scenarios.
6. Handling packets that start with two STP/SDP characters, or ends with two END/EDB characters.

3.1.3.2. Test scenarios as expected input in normal state

Table 43 shows the positive testcases performed to test the Rx functionality and Table 44 show a real scenario to be run.

Table 43: Rx positive test cases

	Feature to be tested	Test case/environment	Expected behavior	Pass/ Fail
1	Handling a sequence of packets with minimum available size (During L0 state)	Consecutive 30 DLLPs	The received packets should be forwarded all to data link layer without any dropping in the data.	Passed
		Consecutive 30 TLPs with minimum size (28 bytes)		Passed
2	Handling a sequence of packets with maximum available size	Consecutive 30 TLPs with maximum size (544 bytes)		Passed
3	Handling TLPs with all available sizes	Consecutive TLPs with all available sizes		Passed
		Consecutive TLPs with all available sizes with DLLPs in between		Passed
4	Handling a sequence of DLLPs after a TLP with maximum size	TLP with maximum size followed by a sequence of DLLPs		Passed
5	Packets coming without resetting the module	Receiving Bytes from PHY without reset the module	Without reset, the module should haven't start to operate yet, so all data (OS or Packets) must be dropped.	passed
6	Reset the module in different scenarios	Between two packets	Resetting the module in different scenarios makes the stored packets or OS flushed, and after the reset signal becomes HIGH again, the Rx should operate again normally.	Passed
		After receiving one packet and in the middle of the second		Passed
		With receiving OS		Passed
7	Handling all different ordered sets	Sequence of TS1s and TS2s	Any incoming order set should be forwarded to the LTSSM.	Passed
		Sequence of TS1s and TS2s and Logical idle in between		Passed
		SKP order sets between two different packets		Passed
8	Activate and deactivate the ACK signal	Deactivate the ACK signal	When the ACK signal is deactivated, the running packet transfer to the data link layer should be finished first. Then the Rx must not start any packet transfer to the data link layer before the ACK signal is activated again.	Passed

9	Disable L0 signal and receive from the PHY layer different types of packets and order sets	Receiving an order set while L0 signal is disabled	When the LTSSM is not in the L0 state yet, so if the received sequence of data represents an OS, it should to be forwarded to the LTSSM. If the received sequence of data represents a packet (TLP or DLLP), it should not to be forwarded neither to the LTSSM nor to the Data link layer.	Passed
		Receiving a packet while L0 signal is disabled		Passed

Table 44: Rx real scenario

TLP 200 Byte	ALL Passed
DLLP 8 Byte	
TLP 32 Byte	
TLP 544 Byte	
Logical Idle	
Logical Idle	
TLP 544 Byte	
DLLP	
SKP	
SKP	
SKP	
Logical Idle	
Logical Idle	
SKP	
TLP 200	
DLLP	

3.1.3.3. Test scenarios which are not expected as an input in normal state:

Pass/fail criteria:

As in PCIe gen1 standard, it is optional to support error handling. In our design the behavior is that some of the errors in the incoming packet sequence are passed to data link layer as it is, and some others are not passed to data link layer and hence all the packet with its error is dropped.As there is no guarantee to solve all the packet errors in the MAC layer, so if our hardware stuck and doesn't operate correctly any more, it will need a hardware reset to start its operation again correctly, and this is considered a fail criteria; if it just do something with this error packet (even wrong or not the best solution) but it operates normally with all next incoming packets, so this is considered a pass criteria.

Pass criteria: the hardware doesn't stuck in the negative test scenario.

Fail criteria: the hardware fails to continue its functionality without reset.

Table 45 shows the negative testcases.

Table 45: Rx negative test cases

Num	Feature to be tested	Test case/ environment	Pass/ Fail	Behavior notes
10	Handling uncompleted packets (packets without end)	Sequence of DLLPs, one of them is received without an END character	Passed	The error packet and the second following pkt to it are stored in the buffer and sent to DLL with errors. The first following packet is flushed and not sent to the Data link layer. In another word, the two following packets also will have issues, then the Rx will return to its right operation.
11	Handling packets without start	Sequence of DLLPs, one of them doesn't start with SDP character	Passed	Any packet that doesn't start with STP or SDP character will not be forwarded to the data link layer.
12	Handling packets with odd number of bytes	Packet with odd number of bytes	Passed	These un expected packets are forwarded to the data link layer but without its indicator (start, end, valid are equal to 0), which makes the data link layer drops it.
13	Handling packets with STP/SDP and END/EBD, but started from second byte	Receiving a packet with its start in the most significant byte, and its end in the least significant byte	Passed	
14	Handling incoming OS in different fault scenarios	Sequence of wrong OS but starts with COM character	Passed	The received OS is forwarded to the LTSSM, as the Rx block doesn't handle errors in OS format.
15	Handling a packet with 2 start characters	Sequence of TLPs, one of them starts with 2 STP characters	Passed	This packet is dropped and the following packet is forwarded to the LTSSM with wrong indicators.
16	Handling a packet with 2 end characters	Sequence of TLPs, one of them ends	Passed	This second END have no impact on the filter decision. All the packets are

		with 2 END characters		forwarded to the data link layer without errors.

3.2. Back-to-Back test

The back-to-back integration is to connect two different MAC layers of two different devices to communicate with each other. As shown in Figure 59, each MAC layer in each device consists of 3 blocks; Tx, Rx, and LTSSM with two different interfaces with the data link layer. The test is done by inserting the input from Data link layer – MAC interface of device [1] then observe the output to the data link layer of device [2] as a primary observation point and the output from the Tx of device [1] as a secondary observation point.

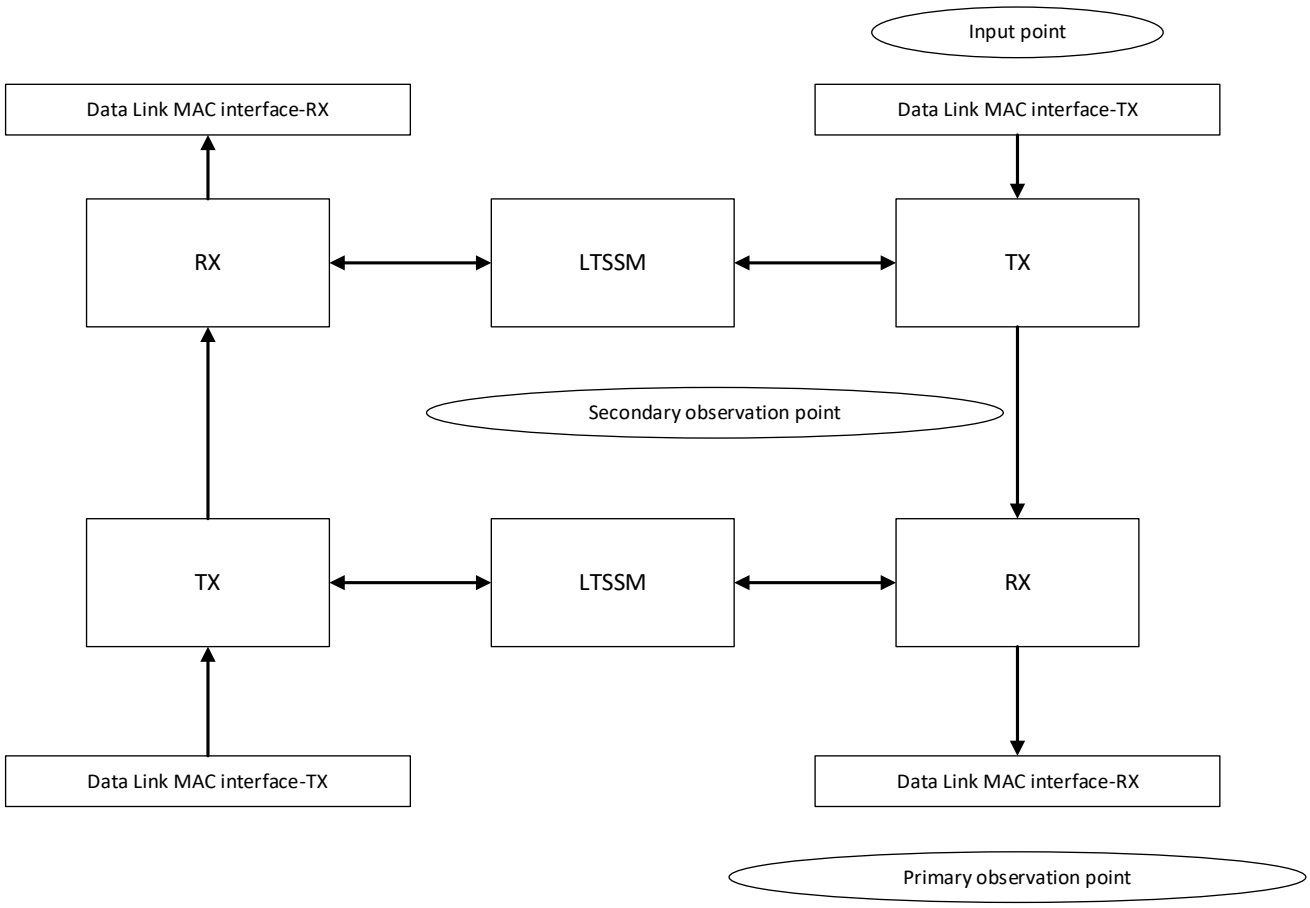

Figure 59: MAC Back-to-Back integration

The objective of this integration is to test the functionality of the MAC layer to make sure that:

1. LTSSM can initialize and re-train the link from detect state to L0 state.
2. Tx can capsulate the incoming packets either TLPs or DLLPs.
3. Tx gives the Ordered set data a priority over TLP and DLLP during L0 state.
4. Tx sends logical Idle if it doesn't receive packets from the data link layer nor ordered sets from LTSSM.
5. Tx shouldn't send any packet during link training and initialization states.
6. Rx can filter the received data and send ordered sets to LTSSM while sending complete packets to the data link layer.
7. Rx should send only one complete packet at a time to the data link layer.

8. Rx shouldn’t send any packets to the Data link layer nor ordered sets to the LTSSM if there is no new data is being received from the PHY layer (Logical Idle), and default values only is put on the buses.
9. Rx shouldn’t send any packet to the Data link layer if the ACK is not enabled.

3.2.1. Pass/Fail Criteria

If the actual behavior meets the standard, then this case is pass. Else, it fails.

3.2.2. Test plan

3.2.2.1. Positive test cases

Table 46 shows the positive testcases performed to test the MAC layer functionality.

Table 46: Back-to-Back positive test cases

#	Feature to be tested	Actual Behavior	Pass/Fail
1	PHY sends the required signals to the LTSSMs of both devices	After the lane is detected, the training sequence starts. So, the negotiation of the TS order set starts. Hence, the transition from the state to another occurs to achieve the L0 state.	Pass
2	Data link layer of device [1] doesn’t send packets while LTSSM is still in link training and initialization, not L0 state.	The output on the lane should be OS data coming from LTSSM.	Pass
3	Rx of device [2] receives OS data while LTSSM is still in link training and initialization, not L0 state.	The incoming OS data received from PHY interface is filtered and then forwarded to LTSSM until the link training has been finished.	Pass
4	Data link layer of device [1] sends packets with different sizes during L0 state.	Data link – MAC interface buffer receives one complete packet, then sends it to Tx buffer. After the whole packet is sent to the Tx buffer, Data link – MAC interface buffer can accept a new packet from the data link layer. The controller acts as the arbiter between Ordered sets data and packets. If the new OS data is received from LTSSM while a packet is being sent on the lane, the packet should be sent till its end, then the OS data is sent through the lane. Else, OS data should be sent first due to its priority.	Pass
5	Rx of device [2] receives OS data and packets with different sizes during L0 state.	If the incoming data starts with COM symbol, in another word if Rx receives OS, all the next bytes are forwarded to LTSSM. Otherwise, the sequence of data which starts with STP or SDP symbols and ends with END symbol, which forms a complete packet, is filtered and the framing symbols are removed then forwarded to	Pass

		Data link layer when the packet is completely received.	

3.2.2.2. Negative test cases

Table 47 shows the negative testcases.

Table 47: B2B - negative testing

Feature to be tested	Expected Behavior	Actual Behavior	Pass/Fail	Note
Data link layer of device [1] sends packets with different sizes while LTSSM is still in link training and initialization, not L0 state.	Data link layer shouldn't send a packet during link training and initialization.	Data link – MAC interface buffer receives one complete packet, then sends it to the Tx buffer. After the whole packet is sent to Tx buffer, Data link – MAC interface buffer can accept a new packet from the data link layer. The output data on the lane should be the ordered sets only.	Fail	This case shouldn't happen in further phases. Data Link Layer should have L0 enable signal to indicate the link is up. Hence, it can send its packets to MAC layer. Otherwise, it shouldn't send any packets.

3.3. IP Integration

3.3.1. Last year IP

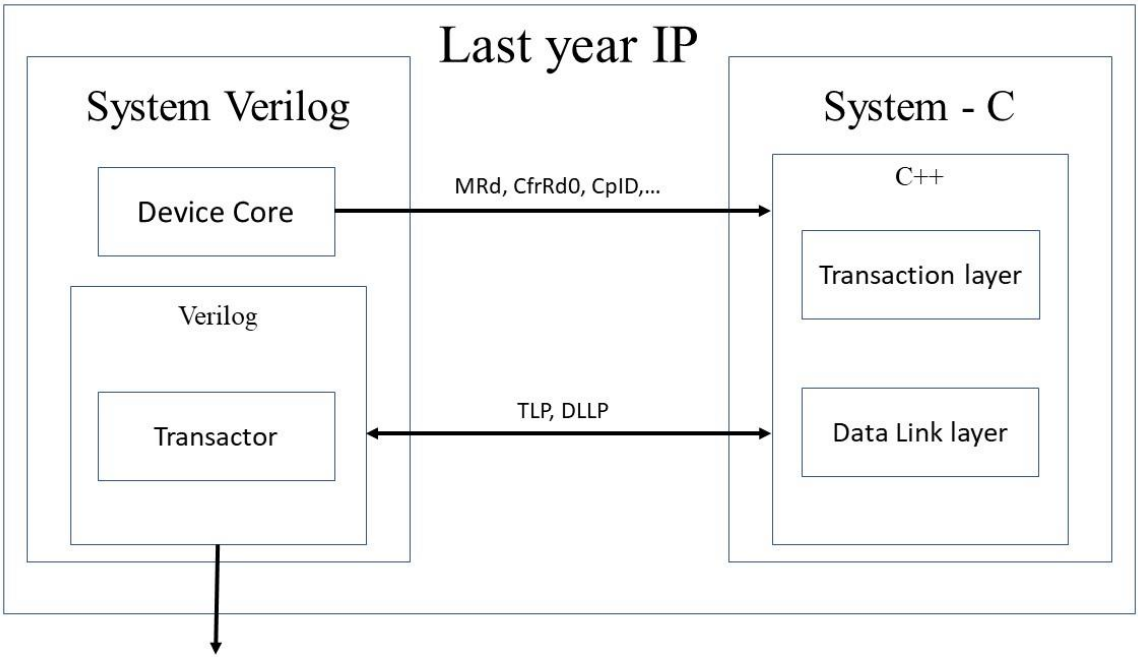

Figure 60: last year IP

The last year project they implement only the transaction layer and data link layer in C++ and replace the physical layer with a pipe that connect two pcie IPs together in verilog. The integration between Verilog and C++ codes was done using DPI in a system Verilog,

The hierarchy of layer in the last year stand-alone IP was as described next:

3.3.1.1. Device Core

As our PCIe has no application layer, they implement the application layer in system Verilog as a test bench. In this test bench, different functions of the transaction layer have been called to generate a traffic of packets.

3.3.1.2. Transaction and Data Link layers

Transaction and data link layers had implemented in C++ and with a system C file to make it is able to call the transaction layer functions from system Verilog test bench. And to run the C++ codes, they used a thread to run the C++ code. The tread runs the codes each 200 clock cycles.

3.3.1.3. Transactor

The transactor was a memory that the transmitter part can write its packet in it. It allows two IPs connect to perform a back-to-back test.

3.3.2. Physical layer integration

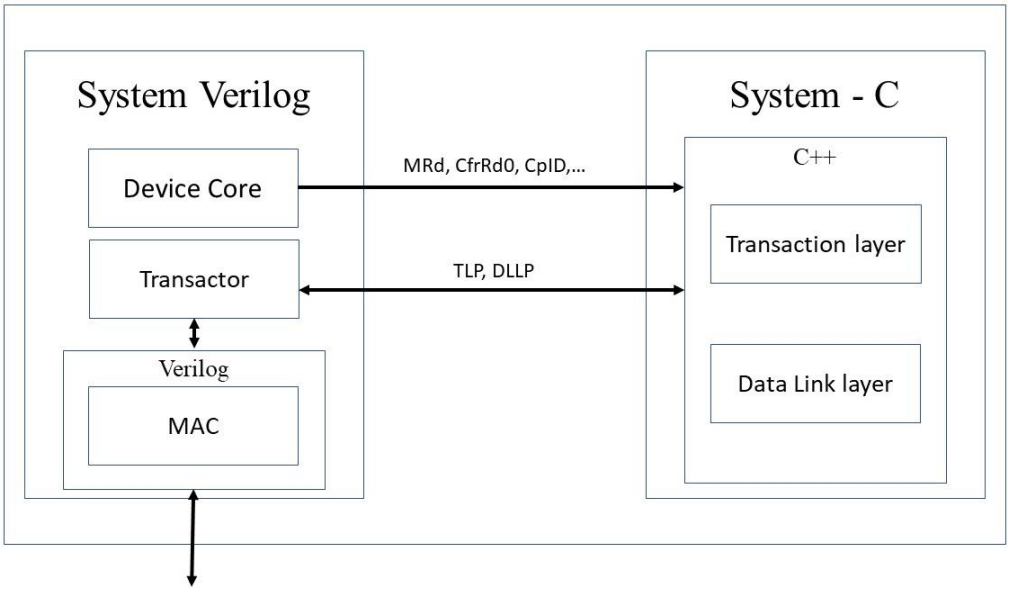

Figure 61: full IP with physical layer

The previous Verilog codes, which described as transactor, is replaced with the logical part of the physical layer-MAC and we define a new interface with system Verilog between Data link layer that implemented in C++ and MAC that implemented in Verilog.

The main objective of the interface layer with the physical layer transmitter part is mapping the incoming packet that stored in array of integers in C++ to a stream of 256 bits each clock cycle with their indicators that illustrated previously in Tx interface with PIPE & LTSSM chapter. And the same thing with the physical layer receiving part, it collects the incoming sequence of bits and store them in array of integer in C++ language, also from the received indicators that illustrated previously in Rx interfaces with PIPE, LTSSM and Data Link Layer chapter it takes decisions about the packet and finally call the Physical_To_DataLink function.

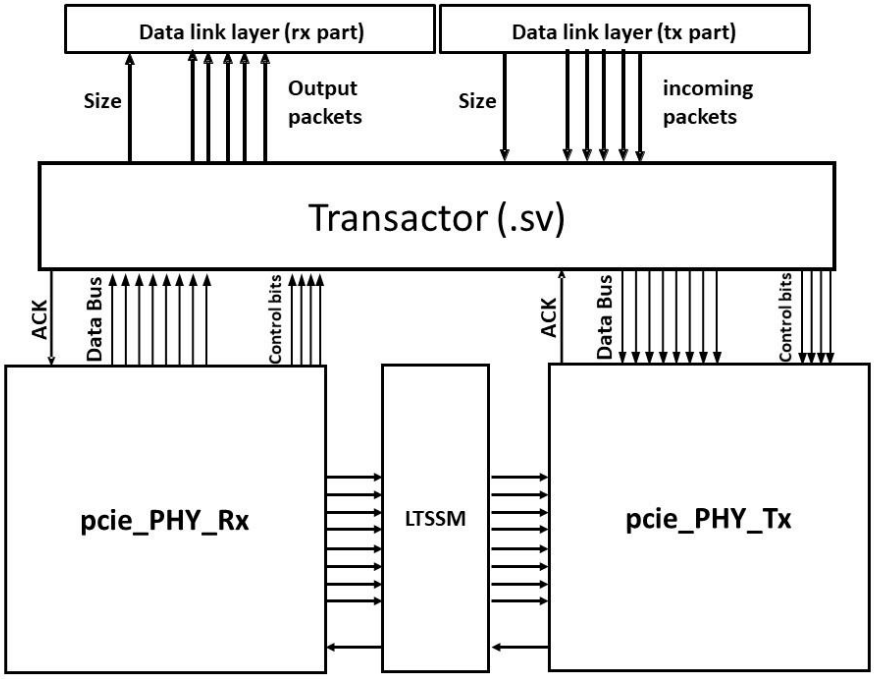

Figure 62: new interface layer between data link layer and physical layer

4. Integration and Testing with industrial core in MENTOR

For more testing, we integrate our IP core that we briefly described in IP Integration chapter with industrial IP mentor core. Integration with Mentor PCIe core, the problems we faced and the final results is discussed in this chapter.

4.1. Mentor PCIe core product

Mentor, A Siemens Business offers a virtual PCIe to test and debug any endpoint like graphics cards, network cards, ...etc.

Virtual PCIe behaves as root complex, Downstream device and need to run a certain application with the help of the target endpoint device. The testing is running on three phases.

- **PCIe Link Training.**

Virtual PCIe runs link training based on configurations. Link training is continued till the two devices reach to the desire link speed.

- **PCIe Enumeration.**

After link training, the root complex requests configuration reads and configuration writes requests from the endpoint device. Implicitly there are flow control packets that are sent by the two devices to negotiate on the available buffer spaces, also ACK/NACK packets and completion packets are sent from them also.

When this process has finished, the root complex is completely recognized the endpoint device and the application that runs over the processor can now runs using the operating system that layered also over the VPCIe.

- **Application Verification.**

This process aims to test the endpoint application layer and it depends only on what this endpoint is designed to do, unlike the previous two phases that runs almost in the same way for all PCIe endpoint devices (depends only on the PCIe-GEN of this targeted endpoint).

In our project we proceed only in the first two phases as we have no application layer to serve.

4.2. Problems between our IP and Mentor IP

Integrating our developed core with an industrial one in MENTOR was challenging because it reveals flaws and error that may not cause a problem when integrating two instances from the developed pcie_1 IP core together. The integration was done by connecting the developed core pcie_1 IP as an endpoint and Mentor's virtual PCIe core as a root complex.

The objective of the integration was to finish link initialization and training step, finish flow control initialization step and finally enumeration step which includes sending and receiving configuration packets. Next, we show the encountered problems and bugs of the developed pcie_1 IP after being successfully complied and linked to mentor's virtual PCIe root complex.

Problem [1]: Stuck in detect state.

The developed IP was designed following version 3.0 of the PIPE standard, which was different from that of the Root Complex

Solution: This was overcome by fixing the sizes of some signal to match that of the RC (o_Rate -> o_Rate [1:0]), and setting others with constant values (o_Tx/RxValid, synchHeader).

Problem [2]: Timeouts at different states

Solution: The timeout values for each state needed to be increased.

Problem [3]: Root Complex (RC) and End point (EP) exchange initialization flow control DLLPs. No one of them send TLP. The code of the last year doesn't contain initialization flow control 2 (InitFC2).

Solution: After revision MindShare, we create (InitFC2) packets and edit **Data_Link_Update** function in **Data_link.cpp** file.

Figure 63 shows the difference between how Root Complex (RC) and EndPoint (EP) exchange initialize flow control before and after editing the data link layer functions.

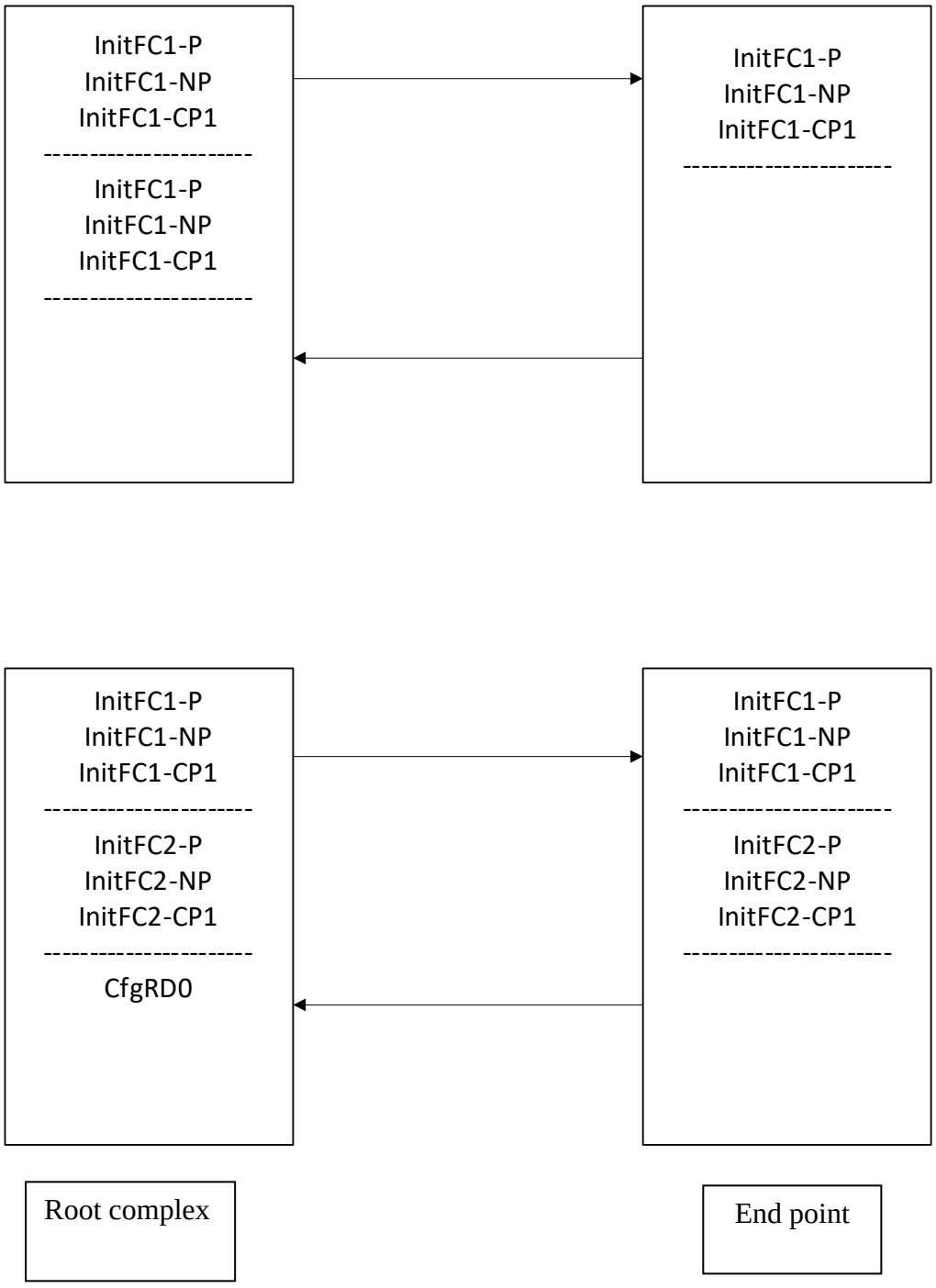

Figure 63: The initial flow control packets exchange before and after editing

Problem [4]: RC sends first TLP. It is Read Configuration (CfgRd0) packet, but our EP doesn't reply with Completion packet (CplD).

In flow control initialization stage, RC sends InitFC-Cpl with HeaderFC = 0 and DataFC = 0 which means it has infinite credits. The Data link layer doesn't handle this case. So, after the creation of CplD in the transaction layer, the data link layer doesn't send it to Physical layer.

Solution: The **Flow_Control_Gating_Logic** function in **Transaction_Layer.cpp** file was edited to handle this case.

Problem [5]: EP sends a TLP packet which RC can't identify. Tx of our Physical layer receives the TLP from the data link layer in a wrong way.

Solution: The **Transactor.sv** file was edited to solve this problem.

Before

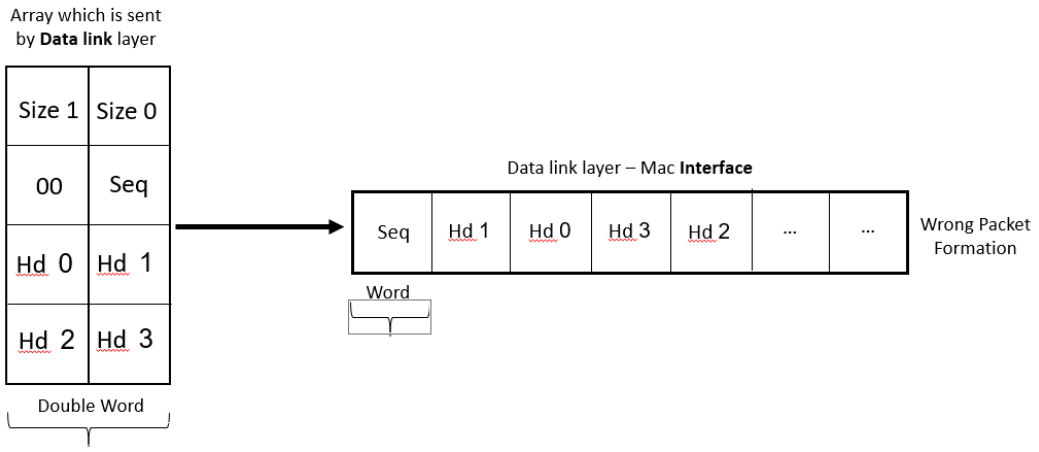

Figure 64: Packet transfer in a wrong way before editing our interface layer

After

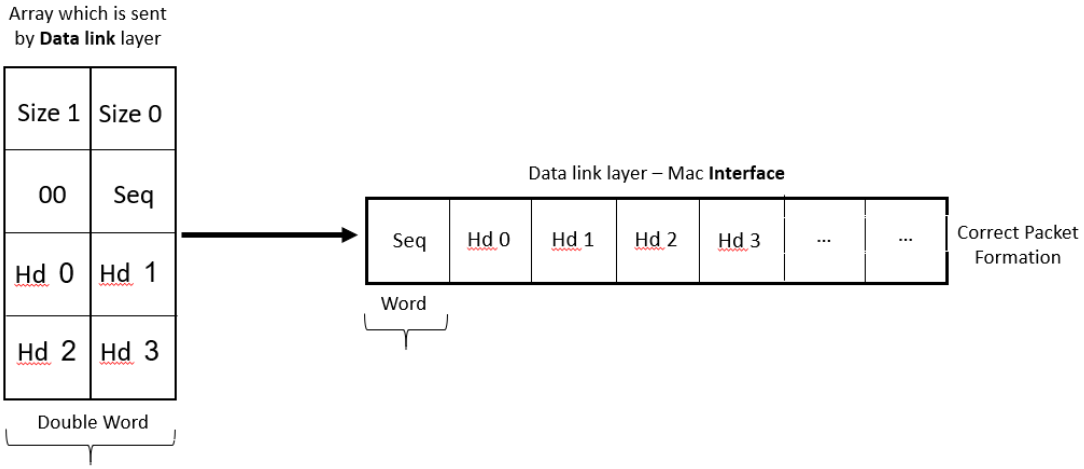

Figure 65: Packet transfer in a right way after editing our interface layer

Problem [6]: RC identifies the received packet as CplD, but it replays with NAK.

The CRC of TLP is implemented in a wrong way. It needs to reverse the order of the incoming data.

Solution: The waveform of example pipe_43 is used to identify the correct value of CRC for specific Header and Data. With try and error approach, the bug was caught. Then, **generate_LCRC_32** function in the **Data_Link.cpp** was edited.

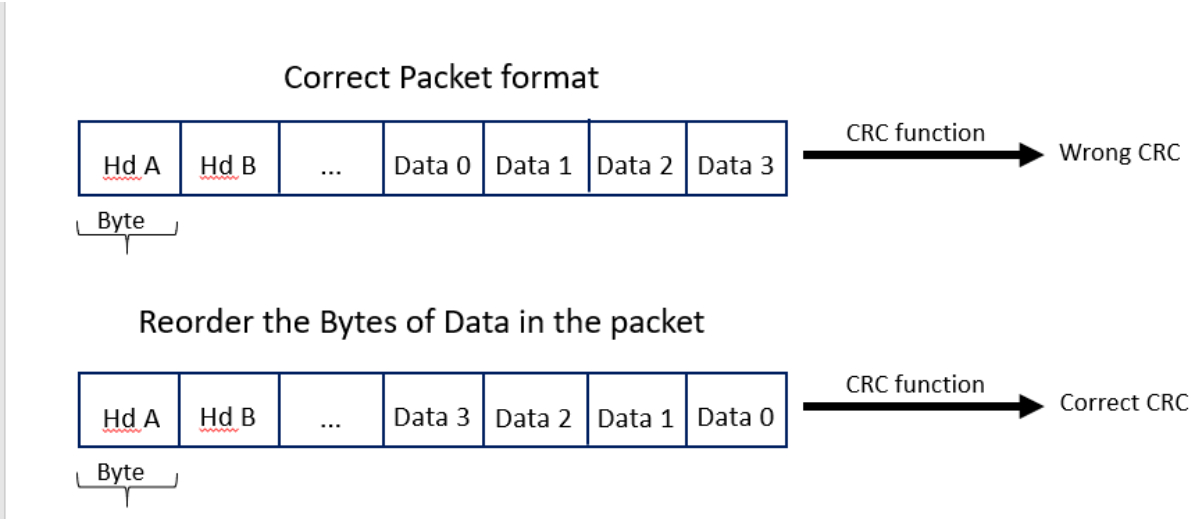

Figure 66: How CRC function order the bytes of the packet before and after editing

Problem [7]: Stuck at recovery state.

During the trails to solve the ACK/NAK problem, one of the conditions to enter recovery state for the RC occurs. As no implementation for recovery state existed in the developing IP.

Solution:A simple recovery state was implemented, which does not take into consideration the link width or speed change options.

5. Future work

In the future, this IP can be modified in some ways. Regarding the physical layer, it can be implemented to support GEN 2 and GEN 3. In addition, the number of lanes can be increased to be 2, 4, 8 or 16. Regarding the transaction layer and the data link layer, they can be implemented as RTL using VHDL or Verilog instead of the modeling with C++. Thus, the full IP can be burned on FPGA to measure the utilization and power. Finally, an application layer which is the device core can be implemented and added to the IP.

6. References

1. MindShare PCI Express System Architecture
2. PHY Interface For the PCI Express and USB 3.0 Architectures Version 3.0
3. PCI Express Base Specification Revision 2.0