Hardware Specification for PCIe Physical Layer Gen5

Prepared by Design Team

January 2021

Contents

1 Introduction **7**
- 1.1 Overview . . . 7
- 1.2 PCIe . . . 7
- 1.3 Lane . . . 8
- 1.4 Generations Speeds . . . 9
- 1.5 Topology . . . 9
- 1.6 Device layers . . . 12
- 1.7 layers interaction . . . 14
- 1.8 TLP packet . . . 15
- 1.9 DLLP packet . . . 15
- 1.10 Physical layer . . . 16
- 1.11 Logical sub-block . . . 16
- 1.12 PIPE Architecture . . . 17
  - 1.12.1 MAC Architecture . . . 17
  - 1.12.2 PHY/MAC Interface . . . 18
  - 1.12.3 PHY Architecture . . . 18
  - 1.12.4 PIPE PLL . . . 18
  - 1.12.5 LPIF Architecture . . . 18

2 Features **20**
- 2.1 Supported Features . . . 20
- 2.2 Supported Blocks . . . 22
  - 2.2.1 TxBuffer . . . 22
  - 2.2.2 Byte Stripping . . . 22
  - 2.2.3 Scrambler . . . 23
  - 2.2.4 Ordered sets . . . 23

3 Signal Interface **24**
- 3.1 PIPE Interface . . . 25
- 3.2 LPIF Inteface . . . 31
- 3.3 Interal Signals . . . 38

4 Architecture **39**
4.1 Abstract Design 39
4.1.1 LTSSM 40
4.1.2 PIPE Interface 41
4.1.3 LPIF Interface 41
4.1.4 Scrambler init 42
4.1.5 Ordered set and packet Generator module 43

5 Scenarios **45**
5.1 PIPE interface scenarios 45
5.1.1 Electrical Idle 45
5.1.2 Link Equalization Evaluation 47
5.1.3 Active PM L0 to L0s and back to L0 49
5.1.4 reestablish block alignment in recovery state 51
5.1.5 RXeqTraining 51
5.1.6 Reset 52
5.1.7 Updating TxDeemph 53
5.1.8 Selecting De-emphasis 54
5.1.9 Signal rate 55
5.1.10 Error Detection 56
5.1.11 8B/10B Decode Errors 56
5.1.12 Disparity Errors 57
5.1.13 Elastic Buffer Errors 58
5.1.14 Receiver detection 60
5.1.15 128b/130b Encoding and Block synchronization 60

List of Figures

1.1 PCIe Link 8
1.2 Topology 10
1.3 switch 11
1.4 configuration software for PCIe 12
1.5 Device A and B connect by Link 13
1.6 Switch port 14
1.7 TLP 15
1.8 DLLP 15
1.9 TLP at data link layer 15
1.10 Physical Layer 16
1.11 PHY/MAC Interface 17
1.12 PLL 18
1.13 LPIF 19

4.1 MAC Layer Architecture 40
4.2 PIPE interface 41
4.3 LPIF interface 42
4.4 Scrambler init 43
4.5 Ordered set and packet Generator module 43

5.1 electrical idle entry for gen1,2 46
5.2 electrical idle entry for gen3 47
5.3 successful link equalization evaluation request 48
5.4 Equalization Evaluation RequestResulting in Invalid Feedback . . 49
5.5 L0s entry for PCIe gen1/2 50
5.6 L0s to L0 transition 50
5.7 reestablish block alignment in recovery state 51
5.8 RXeqTraining 52
5.9 Reset 53
5.10 Updating TxDeemph 54
5.11 Selecting Tx De-emphasis value 54

5.12 Polarity inversion 55
5.13 signal rate 56
5.14 8B/10B Decode Errors 57
5.15 Disparity Errors 58
5.16 Elastic Buffer Underflow 59
5.17 Elastic Buffer Overflow 59
5.18 Receiver detection 60
5.19 128b/130b Encoding and Block synchronization 61

List of Tables

1.1 PCIe Generations Speeds for link width x16 9

2.1 supported states 20
2.2 supported Blocks 21

3.1 TX and Related signals 25
3.2 RX and Related signals 26
3.3 Clk and reset 27
3.4 Commands and Status signals 28
3.5 Message Bus Interface Signals 30
3.6 LPIF Interface Signals 31
3.7 Error signals 34
3.8 Clock Gating Interface from logPHY 34
3.9 Configuration Interface 35
3.10 Signals for PCIe 36
3.11 LTSSM(out/in) and Packet Generator Inteface 38

4.1 Input signals for ordered set module 44

Revision history

Revision Number	Date	Description
0.1	13/1/2021	Initial Draft
1.0	25/1/2021	First resealed of specification

Chapter 1

Introduction

1.1 Overview

PCIe (peripheral component interconnect express) is an interface standard for connecting high-speed components. Every desktop PC motherboard has a number of PCIe slots you can use to add GPUs (aka video cards aka graphics cards), RAID cards, Wi-Fi cards or SSD (solid-state drive) add-on cards. The types of PCIe slots available in the PC will depend on the motherboard that we bought. PCIe slots come in different physical configurations: x1, x4, x8, x16, x32. The number after the x tells how many lanes (how data travels to and from the PCIe card) that PCIe slot has. A PCIe x1 slot has one lane and can move data at one bit per cycle. A PCIe x2 slot has two lanes and can move data at two bits per cycle (and so on).
PCIe x1 card can be inserted into a PCIe x16 slot, but that card will receive less bandwidth. Similarly, a PCIe x8 card can be inserted into a PCIe x4 slot, but it'll only work with half the bandwidth compared to if it was in a PCIe x8 slot.
PCIe has undergone several large and smaller revisions, improving on performance and other features. So there are several generations and in this spec we will design and implement the physical layer of gen 5. The official PCIe 5.0 standard came out in May 2019. It will bring 128 GB/s of throughput. The specification is backwards compatible with previous PCIe generations and also includes new features, including electrical changes to improve signal integrity and backward-compatible CEM connectors for add-in cards. Before designing and implementing the architecture of gen 5, there is an overview to discuss the general basic concepts of PCIe.

1.2 PCIe

Definition 1.2.1 *PCIe is an interface standard for connecting high-speed components. It is a serial bus model.*

There were many problems limiting the performance of the parallel bus:

- Flight time must be less than the clock period or the model won't work.
- Clock skew
- Signal skew

But the serial transport got over these problems for example flight time becomes nonissue as the clock that will latch the data into the receiver is built into the data stream and no external reference clock, so for the same reason no clock skew. Also, signal skew is eliminated within a lane because there is only one bit of data being sent.

1.3 Lane

Definition 1.3.1 *A lane is composed of two differential signaling pairs, one pair for receiving data and the other for transmitting. Thus, each lane is composed of four wires.*

Number of lanes is called "link width" and is represented as x1, x2, x4, x8, x16 and x32. The tradeoff regarding the number of lanes: more lanes increase the bandwidth of the link but it also increases the cost, space requirement and power consumption.

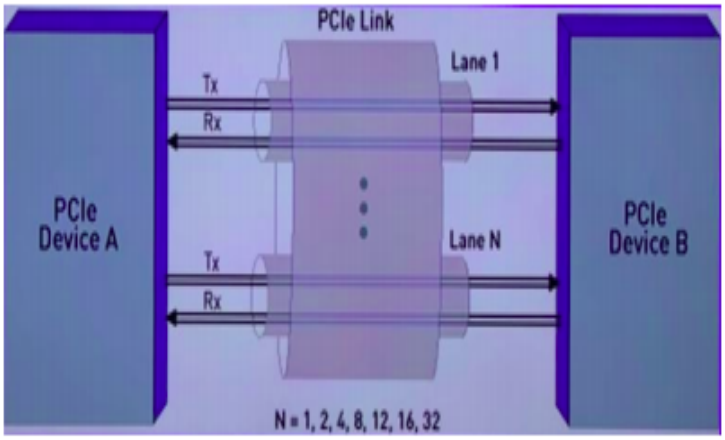

Figure 1.1: PCIe Link

In the receiver, PLL circuit (phase locked loop) takes the incoming bit stream as a reference clock and compares its timing or phase to that of an output clock that it has created with a specified frequency. Based on the result of that comparison, the output clock frequency is increased or decreased until a match is obtained, so the output clock frequency precisely matches the clock that was used to transmit the data.

Each lane uses differential signaling, this improves noise immunity and reduced signal voltage. Moreover, anything that will affect the signal will also affect the other by about the same amount and in the same direction so the receiver won't be affected by the noise that affects the signals and will be able to distinguish the bits.

1.4 Generations Speeds

Table 1.1: PCIe Generations Speeds for link width x16

Version	Bandwidth	Gigatransfer	Frequency
PCIe 1.0	8 GB/s	2.5 GT/s	2.5 GHz
PCIe 2.0	16 GB/s	5 GT/s	5 GHz
PCIe 3.0	32 GB/s	8 GT/s	8 GHz
PCIe 4.0	64 GB/s	16 GT/s	16 GHz
PCIe 5.0	128 GB/s	32 GT/s	32 GHz

Bandwidth calculations for link width x1:

$$PCIe_{B.W_{Gen1}} = (2.5Gb/s \times 2 \quad directions)/10 \quad bits \quad per \quad symbol = 0.5 \quad GB/s \tag{1.1}$$

$$PCIe_{B.W_{Gen2}} = (5Gb/s \times 2 \quad directions)/10 \quad bits \quad per \quad symbol = 1 \quad GB/s \tag{1.2}$$

$$PCIe_{B.W_{Gen3}} = (8Gb/s \times 2 \quad directions)/8 \quad bits \quad per \quad byte = 1 \quad GB/s \tag{1.3}$$

1.5 Topology

A Topology is composed of point-to-point Links that interconnect a set of components. This figure 1.2 illustrates a single fabric instance referred to as a hierarchy – composed of a Root Complex (RC), multiple Endpoints (I/O devices), a Switch, and a PCI Express to PCI/PCI-X Bridge, all interconnected via PCI Express Links.

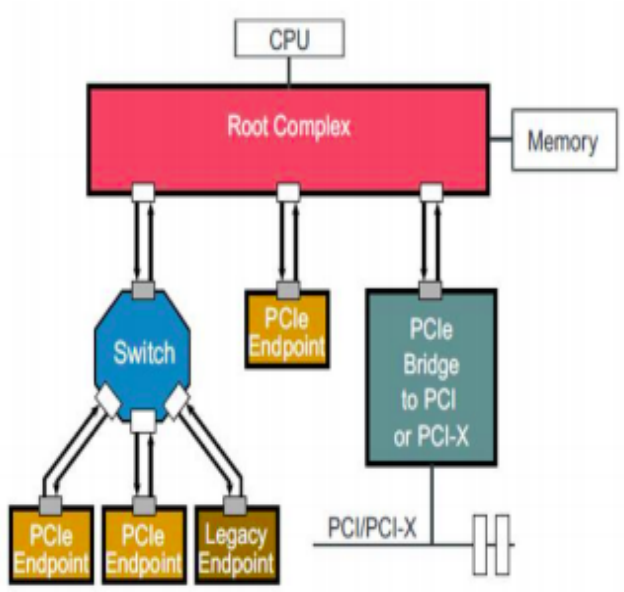

Figure 1.2: Topology

- Root complex: it is the interface between the system CPU and the PCIe topology
  - Root complex may support one or more PCI Express Ports. Each interface defines a separate hierarchy domain. Each hierarchy domain may be composed of a single Endpoint or a sub-hierarchy containing one or more Switch components and Endpoints.
  - The capability to route peer-to-peer transactions between hierarchy domains through a Root Complex is optional and implementation dependent.
- Switch: allows more devices to be attached to a single PCIe port, they act as a packet routers and recognize which path a packet will need to take based on its address or other routing information (may have several downstream ports but only one upstream port).

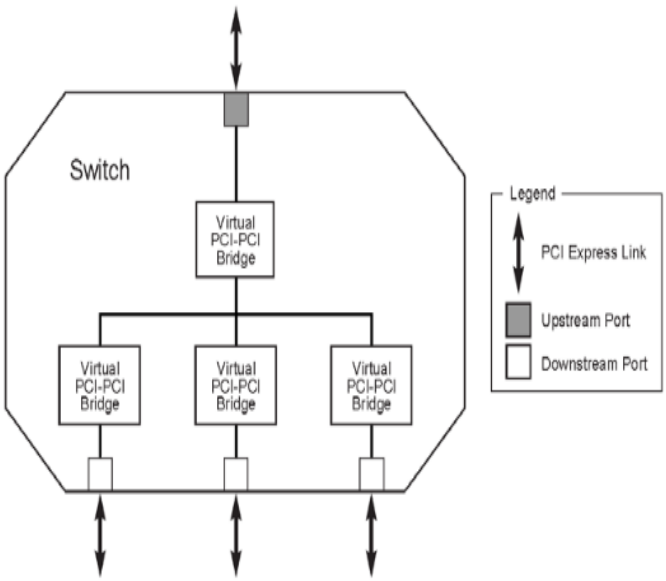

Figure 1.3: switch

All Switches are governed by the following base rules:

- Switches appear to configuration software as two or more logical PCI-to-PCI Bridges.
- Switch must forward all types of Transaction Layer Packets between any set of Ports.
- Switch is not allowed to split a packet into smaller packets.

- Bridge: provides interface to other buses such as PCI or PCI-X or even another PCIe bus.
  Forward bridge $\longrightarrow$ allows older card to be plugged into a new system.
  Reverse bridge $\longrightarrow$ allows a new PCIe card to be plugged into an old PCI system.

- End points: devices that act as initiators and completers of transactions on the bus (they only implement a single upstream port). Endpoints are classified as either legacy, PCI Express, or Root Complex Integrated Endpoints.

Root complex will appear to configuration software as PCI bus number zero and the PCIe ports will appear as PCI to PCI bridges. In a similar way, a switch will appear to software simply as a collection of bridges sharing a common bus. The advantage of this approach is that it allows the transaction routing to take place in the same way it did for PCI.

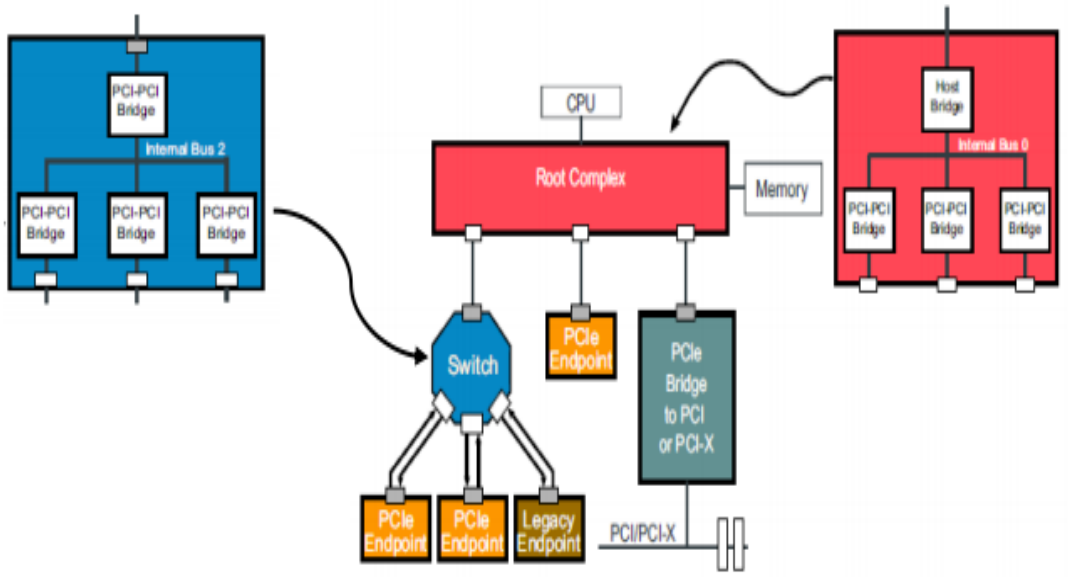

Figure 1.4: configuration software for PCIe

1.6 Device layers

The architecture of PCIe device is divided into three discrete logical layers: Transaction Layer, Data Link Layer and Physical Layer. Each of these layers is divided into two sections: one that processes outbound (to be transmitted) information and one that processes inbound (received) information.

- Transaction layer: creation of transaction layer packet (TLP) on the transmit side and decoding on the receiver side. Also responsible for other 3 functions which are flow control, quality of service and transaction ordering functionality.
- Data link layer: creation of data link layer packet (DLLP) on the transmit side and decoding on the receiver side. Also, responsible for link error detection and correction.
- Physical layer: creation of ordered set packet on transmit side and decoding on the receiver side. It processes all 3 types of packets to be transmitted on the link and processes packets received from the link also. Then packets are encoded and serialized.
- Then link training and status state machine (LTSSM) of the physical layer is responsible for link initialization and training.

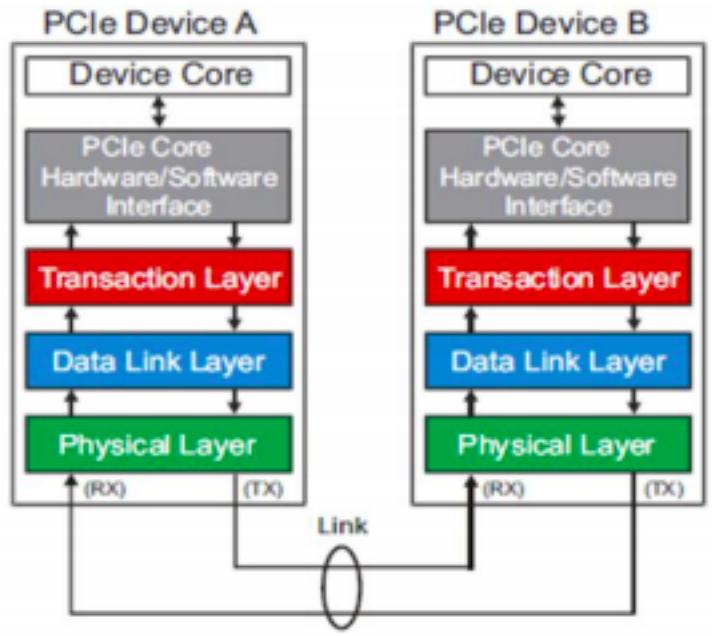

Figure 1.5: Device A and B connect by Link

- Note: switch port needs to implement all the layers as it evaluates the contents of packets, to determine their routing requires looking into the internal details of a packet and that takes place in the transaction layer.

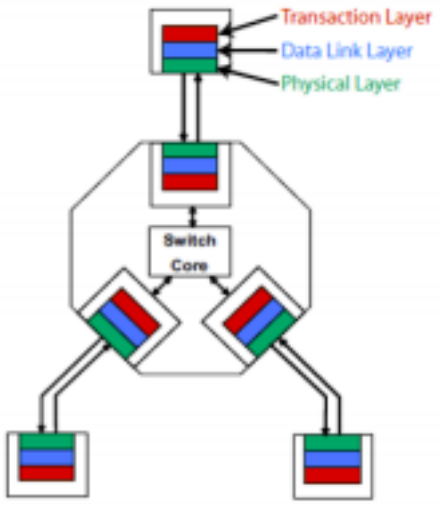

Figure 1.6: Switch port

1.7 layers interaction

- The contents of an outgoing request or completion packet from the device are assembled in the transaction layer based on information presented by device core logic. That information would usually include the type of command desired, the address of the target device and amount of data to transfer.
- The newly created packet is then stored in a buffer called a virtual channel until it is ready for passing to the next layer. When the packet is passed down to data link layer, additional information is added to the packet for error checking at the neighboring receiver and a copy is stored locally so we can send it again if a transmission error occurs.
- When the packet arrives at the physical layer, it is encoded and transmitted differentially using all available lanes of the link.
- When the packet arrives at the receiver, it decodes the incoming bits in the physical layer and check for errors that can be seen at this level.
- if there are no errors, then it forwards the resulting packet up to the data link layer.
- Again the resulting packet is checked, if no errors, then it will be forwarded up to the transaction layer.

- The packet is buffered, checked for errors and disassembled into the original information so the contents can be delivered to the device core of the receiver.

1.8 TLP packet

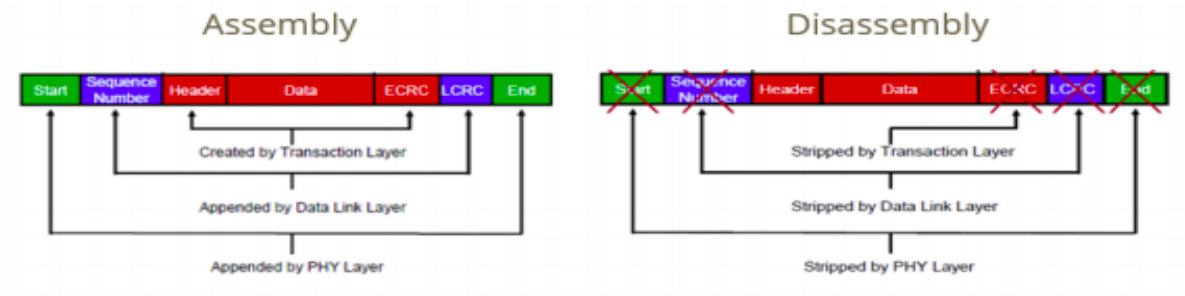

Figure 1.7: TLP

1.9 DLLP packet

They are transferred between data link layers of 2 neighboring devices on a link and the transaction layer isn't aware of these packets. Its size is 8 bytes (very small compared to TLPs).

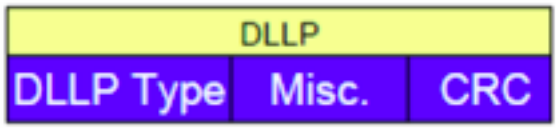

Figure 1.8: DLLP

Note: this DLLP differs from the structure of TLP at the data link layer

Figure 1.9: TLP at data link layer

1.10 Physical layer

It is divided into 2 portions:

1. Logical $\longrightarrow$ contains the digital logic associated with preparing the packets for serial transmission on the link and reversing the process for inbound packets.
2. Electrical $\longrightarrow$ the analog interface that connects to the link and consists of differential drivers and receivers for each lane.

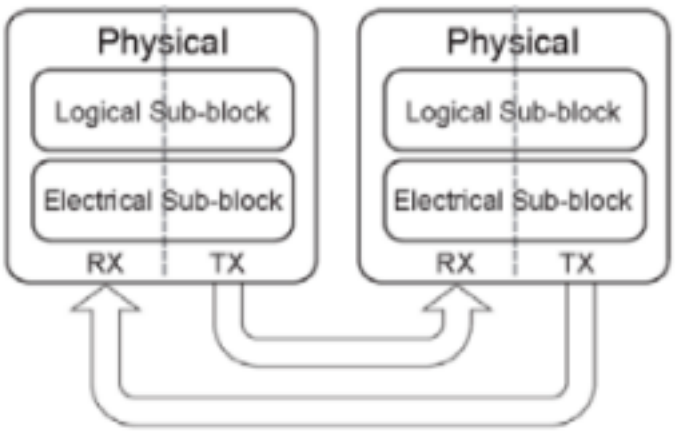

Figure 1.10: Physical Layer

1.11 Logical sub-block

It has two main sections: Transmit section that prepares outgoing information passed from the Data Link Layer for transmission by the electrical sub-block, and Receiver section that identifies and prepares received information before passing it to the Data Link Layer. The logical sub-block and electrical sub-block coordinate the state of each Transceiver through a status and control register interface or functional equivalent. The logical sub-block directs control and management functions of the Physical Layer. PCI Express uses 8b/10b encoding when the data rate is 2.5 GT/s or 5.0 GT/s. For data rates greater than or equal to 8.0 GT/s, it uses a per-lane code along with physical layer encapsulation.

Logical sub-block contains mainly two logical blocks: MAC layer and PHY.

1.12 PIPE Architecture

- Physical Interface for PCI Express Specification (PIPE) developed by Intel, has the stated intent of providing a standard interface between the internal logic of a PCIe design and the analog and high-speed circuitry required to implement the serial link. The purpose of this functional separation is to allow ASIC and integrated circuit designers to focus on the PCI Express device core, Transaction, Data Link and logical Physical Layers, while relying on the PIPE-compliant physical design (PHY) for the electrical interface of the design.
- The PIPE spec defines standard functionality that a PIPE-compliant PHY needs to implement, as well as a standard parallel interface between the PHY and the internal logic referred to in the spec as MAC.

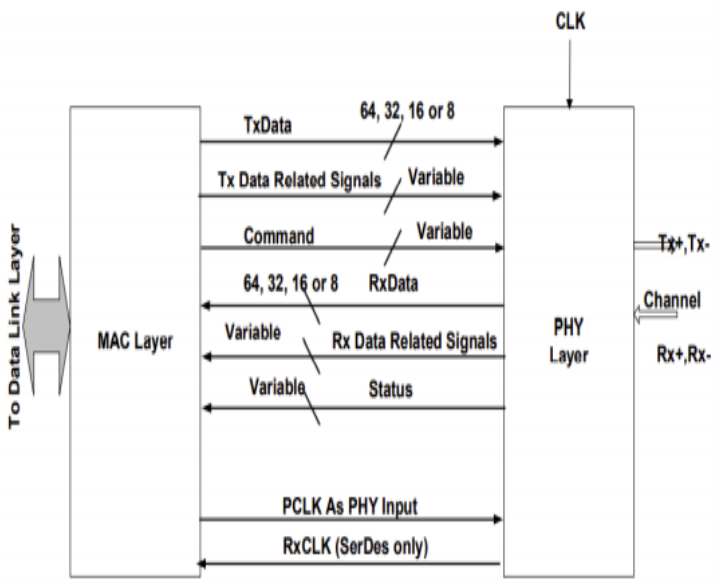

Figure 1.11: PHY/MAC Interface

1.12.1 MAC Architecture

The MAC contains many of the PCIe logical Physical Layer circuits (such as the Link Training and Status State Machine (LTSSM), data scrambling, byte striping and functions as the bridge between the DLL and the PHY/MAC interface)

1.12.2 PHY/MAC Interface

It is a parallel interface for transferring data to be transmitted on the PCIe bus. The width of this parallel interface for bytes of data is shown as either be 8, 16, 32 or 64 bits in each direction.

1.12.3 PHY Architecture

- It contains the 8b/10b encoder and decoder, elastic buffer, serializer and deserializer
- It contains the logic that controls the receiver detection and reports the detect status to the MAC via the PHY/MAC Interface.
- It contains a Phase Lock Loop (PLL) to generate the internal, high speed clocks used for the PHY based on the CLK input.

1.12.4 PIPE PLL

It generates the PCLK used in synchronizing the parallel PHY/MAC Interface based on the CLK input.

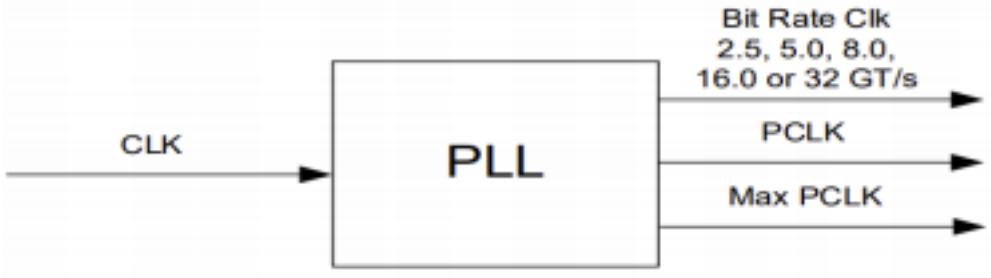

Figure 1.12: PLL

1.12.5 LPIF Architecture

The LPIF specifications defines common interface between the Link Layer and the logical physical layer to facilitate interoperability, design and validation re-use between Link Layers and Physical layers.

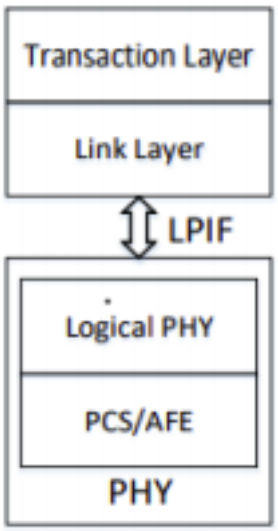

Figure 1.13: LPIF

Chapter 2

Features

In this chapter, we will discuss the blocks and the states that we are going to support, design and implement.

2.1 Supported Features

Table 2.1: supported states

Supported States
Detect
Polling
Configuration
Recovery
L0
L0s

Table 2.2: supported Blocks

Supported Blocks
LPIF error handler
LPIF control
TxBuffer
Ordered sets
LTSSM
Byte Stripping / Unstripping
Scrambler / Descrambler
PIPE register
Arbiter/Mux

The LTSSM is composed of the following 11 states: Detect, Polling, Configuration, Recovery, L0, L0s, L1, L2, Hot Reset, Loopback, Disable. So here is a quick overview about the supported states:

1. Detect
   It is the initial state of the physical layer, only used at Gen1 2.5 GT/s rate, or converted from the data link layer, or after reset, or from other states (Disable, Polling, Configuration, Recovery, etc.) Conversion. In short, the Detect state is the beginning of PCIe link training. In addition, Detect, as the name suggests, needs to implement detection work. Because in this state, the transmitting end TX needs to detect whether the receiving end RX exists and can work normally, if the detection is normal, it can enter other states. The Detect state mainly includes two sub-states: Detect.Quiet and Detect.Active.

2. Polling
   The purpose of this state is to "pair the cipher" and achieve barrier-free communication. After entering this state, the TS1 and TS2 OS sequences are sent between TX and RX to determine Bit Lock, Symbol Lock and solve the problem of Lane polarity reversal. The Polling state mainly includes three sub-states: Polling.Active, Polling.Configuration, Polling.Compliance.

3. Configuration
   The content of this state is very simple. It is to determine the Link/Lane number by sending TS1 and TS2. Configuration contains 6 sub-states: Configuration.LinkWidth.Start, Configuration.LinkWidth.Accpet, Configuration.Lanenum.Wait, Configuration. Lanenum.Accpet, Configuration.Complete, Configuration.Idle

4. Recovery

When the PCIe link needs to be retrained, it enters the Recovery state. There are mainly the following situations:

- When the PCIe link signal finds an error, the Bit Lock and Symbol Lock need to be adjusted
- Exit from L0s state
- Speed Change. Because the first time you enter the L0 state, the rate is 2.5GT/s. When you need to adjust the rate to 5.0GT/s or 8.0GT/s, you need to enter the Recovery state for Speed Change. At this stage, Bit Lock, Symbol Lock, etc. Need to reacquire
- Need to readjust the Width of PCIe link
- Need to readjust the Width of PCIe link
- Only in Gen3 and Gen4, Equalization needs to be performed again.

5. L0
   This is the normal state (Normal and Full-Active State) of the link, and all TLP, DLLP and Ordered Sets can be sent and received normally. In this state, the rate can be 2.5GT/s or 5GT/s or higher (if the devices at both ends of the link support it and have undergone Re-Training).

6. L0s
   The ASPM (active-state power management) state is mainly used to reduce power consumption, and can enter this state when the bus is idle, and can quickly switch back to the L0 state from this state. When in the L0 state, when EIOS appears on the link, it means that it is about to enter the L0s state. When in the L0s state, when FTS appears on the link, the link will quickly complete bit lock and symbol lock, and enter the L0 state.

2.2 Supported Blocks

2.2.1 TxBuffer

The ASPM (active-state power management) state is mainly used to reduce power consumption, and can enter this state when the bus is idle, and can quickly switch back to the L0 state from this state. When in the L0 state, when EIOS appears on the link, it means that it is about to enter the L0s state. When in the L0s state, when FTS appears on the link, the link will quickly complete bit lock and symbol lock, and enter the L0 state.

2.2.2 Byte Stripping

Bytes are striped across multiple lanes $\longrightarrow$ distribution of each byte to different lanes in turn to avoid different lanes with different data lengths.

2.2.3 Scrambler

PCIe uses pseudo random data scrambling to spread off the RF energy in the frequency spectrum $\longrightarrow$ less electromagnetic interference (EMI). This is done by using a linear feedback shift registers on both the sender and receiver side. lfsr maintains an internal state machine which is the top flip-flops. An XOR operation is done between the state of the lfsr and the data $\longrightarrow$ resulting in a predictable and repeatable sequence of pseudo random data. A parallel Scrambler performs 2 things: advance and apply. Advance means that the pattern changes, while apply means the XORing operation performed with this pattern.

2.2.4 Ordered sets

They are physical layer packets that only exists between Tx and Rx's physical layer.

- TS1OS and TS2OS (Training Sequence 1 and 2)
- FTSOS (Fast Training Sequence)
- EIEOS (Electrical Idle Exit)
- EIOS (Electrical Idle)
- SOS (SKIP)

For more details about each block and how all the blocks interact together, see chapter 4.

Chapter 3

Signal Interface

3.1 PIPE Interface

Table 3.1: TX and Related signals

Name	Active Level	Description
TxData[7:0]	N/A	Parallel data input bus.
TxDataValid	N/A	This signal allows the MAC to instruct the PHY to ignore the data interface for one clock cycle. A value of one indicates the phy will use the data, a value of zero indicates the phy will not use the data.
TxElecIdle	High	Forces Tx output to electrical idle when asserted except in loopback. Note: The MAC must always have TxDataValid asserted when TxElecIdle transitions to either asserted or de-asserted; TxDataValid is a qualifier for TxElecIdle sampling.
TxDataK	N/A	A value of zero indicates a data byte, a value of 1 indicates a control byte.
TxStartBlock	N/A	This signals allow the MAC to tell the PHY the starting byte for a 128b block when value is 1. The starting byte for a 128b block must always start with byte 0 of the data interface.
TxSyncHeader[1:0]	N/A	Provides the sync header for the PHY to use in the next 130b block. 01 $\longrightarrow$control byte 10 $\longrightarrow$data byte
TxDetectRx/ Loopback	High	Used to tell the PHY to begin a receiver detection operation or to begin loopback

Table 3.2: RX and Related signals

<table>
<tr><th>Name</th><th>Active Level</th><th>Description</th></tr>
<tr><td>RxData[7:0]</td><td>N/A</td><td>Parallel data output bus.</td></tr>
<tr><td>RxDataValid</td><td>N/A</td><td>This signal allows the PHY to instruct the MAC to ignore the data interface for one clock cycle. A value of one indicates the mac will use the data, a value of zero indicates the mac will not use the data.</td></tr>
<tr><td>RxElecIdle</td><td>High</td><td>Indicates receiver detection of an electrical idle. While deasserted with the PHY in P2 (PCI Express mode).</td></tr>
<tr><td>RxDataK</td><td>N/A</td><td>A value of zero indicates a data byte, a value of 1 indicates a control byte.</td></tr>
<tr><td>RxStartBlock</td><td>N/A</td><td>This signals allow the PHY to tell the MAC the starting byte for a 128b block when value is 1. The starting byte for a 128b block must always start with byte 0 of the data interface.</td></tr>
<tr><td>RxSyncHeader[1:0]</td><td>N/A</td><td>Provides the sync header for the MAC to use in the next 130b block.

01 $\longrightarrow$control byte
10 $\longrightarrow$data byte</td></tr>
<tr><td>RxValid</td><td>High</td><td>Indicates symbol lock and valid data on RxData and RxDataK and further qualifies RxDataValid when used.</td></tr>
<tr><td>RxStatus[2:0]</td><td>N/A</td><td>Encodes receiver status and error codes for the received data stream when receiving data.
<table>
<tr><th>[0]</th><th>[1]</th><th>[2]</th><th>Description</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Received data OK</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1 SKP added</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1 SKP removed</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Receiver detected</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Both $8B/10B$ ($128B/130B6$) decode error and (optionally) Receive Disparity error</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Elastic Buffer overflow</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Elastic Buffer underflow.</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Receive disparity error (Reserved if Receive Disparity error is reported with code 0b100)</td></tr>
</table>
</td></tr>
</table>

RxStandby	Low	Controls whether the PHY RX is active when the PHY is in P0 or P0s. 0 $\longrightarrow$ *Active* 1 $\longrightarrow$ *Standby*
RxStandbyStatus	Low	The PHY uses this signal to indicate its RxStandby state. 0 $\longrightarrow$ *Active* 1 $\longrightarrow$ *Standby*

Table 3.3: Clk and reset

Name	Active Level	Description
Reset#	Low	Resets the transmitter and receiver. This signal is asynchronous.
CLK	Edge	This differential Input is used to generate the bit-rate clock for the PHY transmitter and receiver.
PCLK	Rising Edge	All data movement across the parallel interface is synchronized to this clock.

Table 3.4: Commands and Status signals

<table>
<tr><th>Name</th><th>Active Level</th><th>Description</th></tr>
<tr><td>PowerDown[3:0]
M2P</td><td>N/A</td><td>Power up or down the transceiver. Power states PCI Express Mode:
<table>
<tr><th>[0]</th><th>[1]</th><th>[2]</th><th>[3]</th><th>Description</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>P0, normal operation</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>P0s, low recovery time latency, power saving state</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>P2, lowest power state</td></tr>
</table></td></tr>
<tr><td>Rate[3:0]
M2P</td><td>N/A</td><td>Control the link signaling rate.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>Use 2.5 GT/s signaling rate</td></tr>
<tr><td>1</td><td>Use 5 GT/s signaling rate</td></tr>
<tr><td>2</td><td>Use 8 GT/s signaling rate</td></tr>
<tr><td>3</td><td>Use 16 GT/s signaling rate</td></tr>
<tr><td>4</td><td>Use 32 GT/s signaling rate</td></tr>
</table></td></tr>
<tr><td>PHY Mode[3:0]
M2P</td><td>N/A</td><td>PHYMode[1:0]

$0 \longrightarrow PCIExpress$</td></tr>
</table>

<table>
<tr><td>PhyStatus
P2M</td><td>High</td><td>Used to communicate completion of several PHY functions including stable PCLK and/or Max PCLK (depending on clocking mode) after Reset# deassertion, power management state transitions, rate change, and receiver detection.</td></tr>
<tr><td>Width[1:0]</td><td>N/A</td><td>Controls the PIPE data path width.<table><tr><th>Value</th><th>Datapath Width</th></tr><tr><td>0</td><td>8 bits</td></tr><tr><td>1</td><td>16 bits</td></tr><tr><td>2</td><td>32 bits</td></tr></table></td></tr>
<tr><td>PCLK Rate[4:0]</td><td>N/A</td><td>Control the PIPE PCLK rate.<table><tr><th>Value</th><th>clk rate</th></tr><tr><td>0</td><td>32.25 Mhz</td></tr><tr><td>1</td><td>62.5 Mhz</td></tr><tr><td>2</td><td>125 Mhz</td></tr><tr><td>3</td><td>250 Mhz</td></tr><tr><td>4</td><td>500 Mhz</td></tr><tr><td>5</td><td>1000 Mhz</td></tr><tr><td>6</td><td>2000 Mhz</td></tr><tr><td>7</td><td>4000 Mhz</td></tr></table></td></tr>
<tr><td>PclkChangeAck
M2P</td><td>High</td><td>Only used when PCLK is a PHY input. Asserted by the MAC when a PCLK rate change or rate change or, if required, width change is complete and stable.

After the MAC asserts PclkChangeAck the PHY responds by asserting PhyStatus for one cyle and deasserts PclkChangeOk at the same time as PhyStatus. The controller shall deassert PclkChangeAck when PclkChangeOk is sampled low.</td></tr>
<tr><td>PclkChangeOk
P2M</td><td>High</td><td>Only used when PCLK is a PHY input. Asserted by the PHY when it is ready for the MAC to change the PCLK rate or Rate.</td></tr>
</table>

Table 3.5: Message Bus Interface Signals

Name	Direction	Description
M2P_MessageBus [7:0]	Input	The MAC multiplexes command, any required address, and any required data for sending read and write requests to access PHY PIPE registers and for sending read completion responses and write ack responses to PHY initiated requests.
P2M_MessageBus [7:0]	Output	The PHY multiplexes command, any required address, and any required data for sending read and write requests to access MAC PIPE registers and for sending read completion responses and write ack responses to MAC initiated requests.

3.2 LPIF Inteface

Table 3.6: LPIF Interface Signals

Name	Active Level	Description
lclk	High	Link Clock: The clock frequency the LPIF interface operates at. The Link Clock is an input to signal to both the Link Layer as well as the Logical PHY.
pl_trdy	High	LIndicates Physical Layer is ready to accept data. Data is accepted when pl_trdy, lp_valid, and lp_irdy are asserted together.
lp_irdy	High	Link Layer to Physical Layer indicating Link Layer is ready to transfer data. lp_irdy must not be presented by the upper layers when pl_state_sts is RESET.
lp_valid [LP_NVLD-1:0]	High	Link Layer to Physical Layer indicates data valid on the corresponding lp_data bytes. 'LP_NVLD' equals the number of valid bits. The bytes of lp_data associated with a specific bit of lp_valid is implementation specific. When lp_irdy is asserted, at least one of the bits of lp_valid must be asserted.
pl_data [NBYTES-1:0][7:0]	N/A	Physical Layer to Link Layer Data, where 'NBYTES' equals number of bytes determined by the supported data bus for the LPIF interface.
pl_valid [PL_NVLD-1:0]	High	Physical Layer to Link Layer indicates data valid on pl_data. 'PL_NVLD' equals the number of valid bits. The bytes of pl_data associated with a specific bit of pl_valid is implementation specific.
pl_stallreq	High	Physical Layer request to Link Layer to flush all packets for state transition
lp_data [NBYTES-1:0][7:0]	N/A	Link Layer to Physical Layer Data, where 'NBYTES' equals number of bytes determined by the data width for the LPIF instance.

lp_stallack	High	Link Layer to Physical layer indicates that the packets are aligned (if pl_stallreq was asserted) and logPHY may begin state transitions.
lp_state_req[3:0]	N/A	Link Layer Request to Logical Physical Layer to request state change. Encodings as follows: 0000: NOP 0001: Active 0010: Active.L0s 0011: Deepest Allowable PM State [L1 Substates only] 0100: L1.1 0101: L1.2 0110: L1.3 0111: L1.4 1000: L2 1001: LinkReset 1010: Reserved 1011: Retrain 1100: Disable
pl_state_sts[3:0]	N/A	Physical Layer to Link Layer Status indication of the Interface. Encodings as follows: 0000: NOP 0001: Active 0010: Active.L0s 0011: Deepest Allowable PM State [L1 Substates only] 0100: L1.1 0101: L1.2 0110: L1.3 0111: L1.4 1000: L2 1001: LinkReset 1010: Reserved 1011: Retrain 1100: Disable
pl_lnk_cfg[2:0]	N/A	Width of the Port: This bit field indicates the width of the port as determined by the Link initialization: 000 – x1 001 – x2 010 – x4 011 – x8 100 – x12 101 – x16 110 – x32

pl_rxframe_errmask	High	Rx Framing Error Reporting Mask: When asserted, receiver framing error logging/escalation should be masked off in the Link Layer. logPHY asserts this based on link state and data path alignment to make sure false errors are not logged by the Link Layer.
pl_speedmode[2:0]	N/A	Current Link Speed as negotiated by the logPHY (3'b000=Gen1,3'b001=Gen2,3'b010=Gen3, 3'b011=Gen4, 3'b100=Gen5, rest=Rsvd) Link Layer should only consider this to be relevant when pl_state_sts=RETRAIN or ACTIVE.
pl_setlabs	High	logPHY's pulsed indication to set Link Auto Bandwidth Change status in Link Status register
pl_protocol[2:0]	N/A	logPHY indication to upper layers about which protocol was detected during training. It has the following encodings: 000b – PCIe
pl_protocol_vld	High	Indication that pl_protocol has valid information. This is a level signal, asserted when the logPHY has discovered the appropriate protocol, but can deassert again after subsequent transitions to RESET state depending on the link state machine transitions.
lp_force_detect	High	This is a level signal. It forces logPHY to shut down the receiver, drive and keep the physical LTSSM in Detect.
pl_phyinrecenter	High	Physical Layer to Link Layer indication that the Physical Layer is in Recovery (Retrain) state. Please note that pl_state_sts indicates the status of the transmitter (training sequence sent by logPHY for example) whereas this signal is asserted when the receiver detects recovery entry (training sequences received by the logPHY for example)

Table 3.7: Error signals

Name	Active Level	Description
pl_error	High	Indicates that the Physical layer detected an encoding or framing related error. This signal shall be asserted by the Logical PHY when non training related errors are detected. Please note that non-training errors are logical PHY specific. The logging of errors must be determined by the upper level protocols.
pl_trainerror	High	Indicates that Physical layer training. Please note that training errors are logical PHY specific. The logging of errors must be determined by the upper level protocols. Logical PHY may use this signal to indicate other uncorrectable errors as well (such as internal parity errors) and transition to LinkError state as a result.
pl_cerror	High	Indicates that Physical Layer received an error which was corrected by Physical Layer. The logging of errors must be determined by upper layer protocols.
lp_linkerror	High	Link Layer to Physical Layer indication that an uncorrectable error has occurred and Physical Layer must move to LinkError State when it samples this signal.

Table 3.8: Clock Gating Interface from logPHY

Name	Active Level	Description
pl_exit_cg_req	High	When asserted, requests upper layers to exit clock gated state as soon as possible.
lp_exit_cg_ack	High	When asserted, indicates that upper layers are not in clock gated state and are ready to receive packets from the Physical Layer.

Table 3.9: Configuration Interface

Name	Active Level	Description
pl_cfg[NC-1:0]	N/A	This is the configuration interface from Logical PHY to the Link Layer.
pl_cfg_vld	High	When asserted, indicates that pl_cfg has valid information that should be consumed by the Link Layer. receive packets from the Physical Layer.
lp_cfg[NC-1:0]	N/A	This is the configuration interface from Link Layer to Logical PHY.
lp_cfg_vld	High	When asserted, indicates that lp_cfg has valid information that should be consumed by the Logical PHY.

Table 3.10: Signals for PCIe

Name	Active Level	Description
pl_nbstallreq	High	Physical Layer request to Link Layer to align packets at LPIF width boundary
lp_nbstallack	High	Link Layer acknowledge to pl_nbstallreq.
pl_block_dl_init	High	Indication from the logPHY to the LL to block initialization of DLLPs.
lp_dl_active	High	Indication from the LL that Data Link Control and Management State Machine is in DL_Active state (as defined in the PCIe spec)
lp_good_dllp	High	ndication from Link Layer that a Data Link Layer Packet (DLLP) was received without errors. Used by upstream ports to block DLLP transmission until a good DLLP is received
pl_in_rxl0s	N/A	Indication from logPHY that Receiver is in L0s
pl_byte_err[(n-1):0]	N/A	Corresponding byte of data has an error. In Gen1/2 framing, it denotes k-char error detected by logPHY. LL uses this to log an error. It may assert in higher data rates, but it is not logged by the Link Layer for Gen3 and above speeds.
	N/A	Corresponding byte of data has an error. In Gen1/2 framing, it denotes k-char error detected by logPHY. LL uses this to log an error. It may assert in higher data rates, but it is not logged by the Link Layer for Gen3 and above speeds.
pl_kchar[(n-1):0]	N/A	k-char indication from logPHY. When asserted, the corresponding data byte must be interpreted at the LL as a k-char.

pl_dlpstart[w-1]	N/A	logPHY indicates the start of a Data Link Layer packet for Gen3 and above speeds. Each bit corresponds to a specific data byte depending on the configuration of the port.
pl_dlpend[w-1]	N/A	logPHY indicating the end of a Data Link Layer packet for Gen3 and above speeds. Each bit corresponds to a specific data byte depending on the configuration of the port.
pl_tlpstart[w-1]	N/A	logPHY indicating the start of a Transaction Layer packet for Gen3 and above speeds (STP).
pl_tlpend[w-1]	N/A	logPHY indicating the end of a Transaction Layer packet for Gen3 and above speeds (END).
pl_tlpedb[w-1]	N/A	logPHY indicating EDB received for Gen3 and above speeds.
pl_rx_flush	N/A	Request from logPHY to Link Layer to flush its receiver pipeline. This typically occurs for framing errors in Gen3

3.3 Interal Signals

Table 3.11: LTSSM(out/in) and Packet Generator Inteface

Name	In/Out	Description
ordered_set_type [3:0]	b	c d h
Gen_type	out	c d h
lane_numbers [3:0]	out	c d h
link_numbers	out	c d h
data_rate [2:0]	out	c d h
Auto_change	out	c d h
speed_change	out	c d h

Chapter 4

Architecture

4.1 Abstract Design

Mac Layer resides between link and physical layers which LPIF interface provides connection between mac and link layer to know this signals see Section 3.2 and PIPE interface provide connection between mac and PCS layer see to know signals see Section 3.1.

We divide design into several modules to reduce functionality of each module so, we seperate PIPE module from LTSSM that his responsible for interacting with PIPE signals and send respond signal to LTSSM. Making isolation to LTSSM from any outside signals is best choice for good design.
Mac Layer modules:

- LPIF Interface
- PIPE Interface
- LTSSM
- PIPE Register
- Packet Generator
- TX and RX Buffers
- Byte Stripping and Byte un-strpping
- Scrambler and De-Scrambler
- Scrambler init
- Aribiter/Mux
- Packet filtering

- ordered sets module

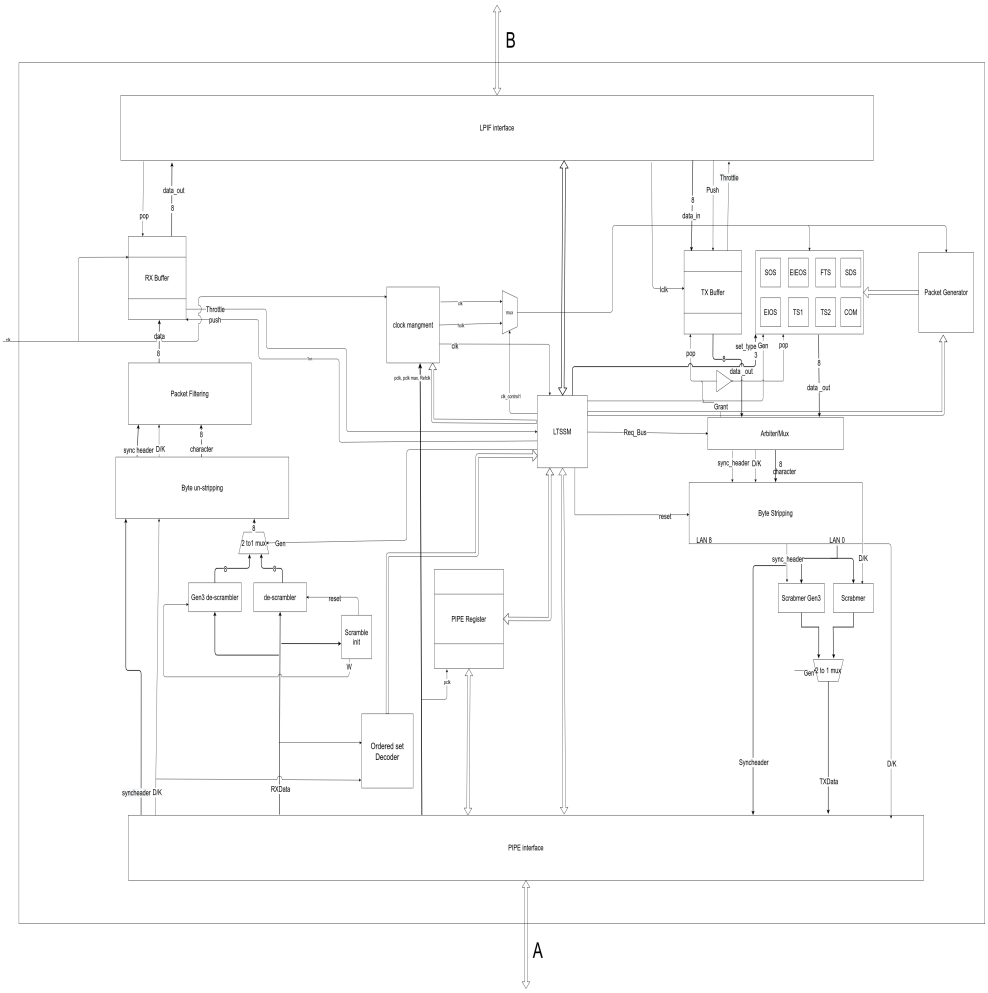

Figure 4.1: MAC Layer Architecture

4.1.1 LTSSM

- Contain Active State Power Management which responsible for power states of the device.
- change link width and bit rate.
- send parameters(lane and link number, ...) to Generate Packet module to generate required ordered sets.

- send Req_Bus signal to arbiter/mux to choose between ordered sets or packet symbol.
- choose PCIe generation mode.

4.1.2 PIPE Interface

It Contains 4 main modules each:

- Rx module
  It receives data from PCS layer in signal RXData and contains Rx related signals(RXsybcheader, RXDatak, ...) see Table 3.2
- Tx module
  It tramsites data to PCS layer in signal TXData and contains Tx related signals(TXsybcheader, TXDatak, ...) see Table 3.1
- Error handler
  It receives all error signals and generate responde to LTSSM Note: signal details in Section 3.1

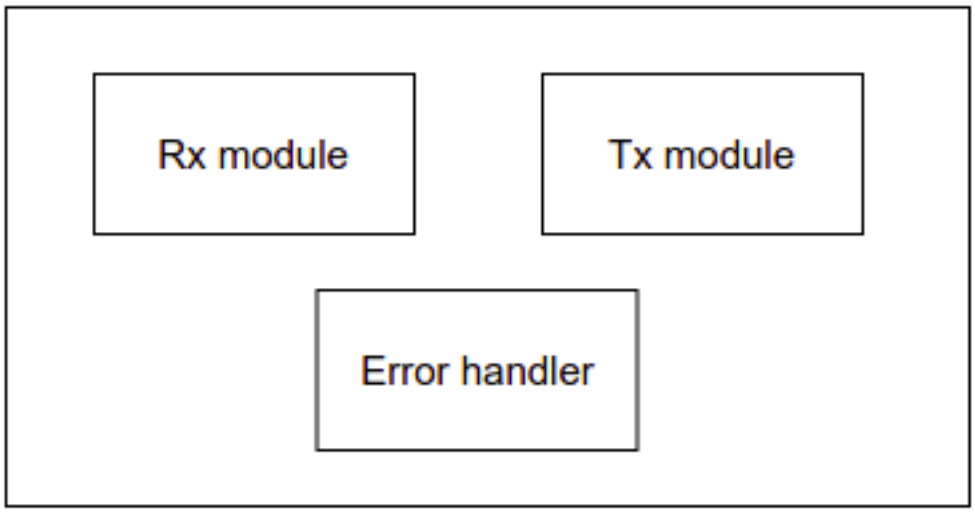

Figure 4.2: PIPE interface

4.1.3 LPIF Interface

- LPIF Error handler receives all error signals then make some process to send decision to LPIF ctl

- LPIF ctl control Txbuffer and Rxbuffer based on error and status signals and send his status to LTSSM to control other modules based on this situation

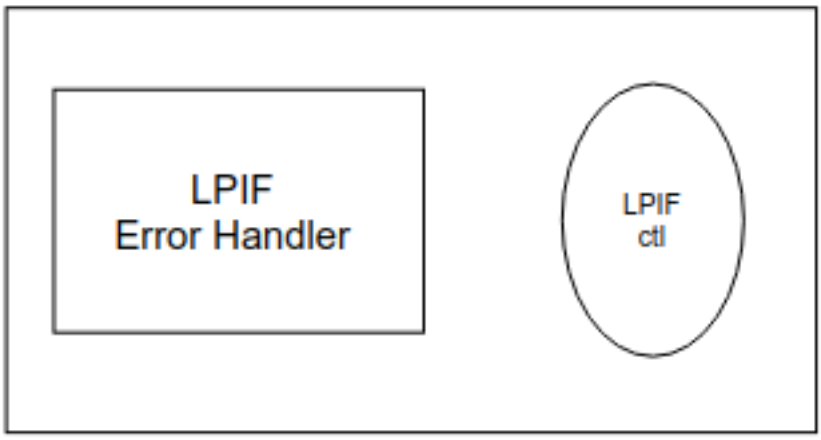

Figure 4.3: LPIF interface

4.1.4 Scrambler init

- COM Detector
  Detect 6 COM symbols to reset scrambler in gen 1,2
- SKP Detector
  Detect skp ordered set based on this it sets W signal which next value LFSR is taked as a seed to scrambler gen 3

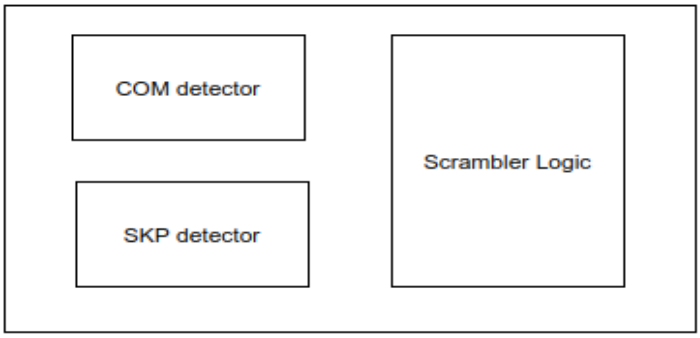

Figure 4.4: Scrambler init

4.1.5 Ordered set and packet Generator module

- Ordered Set module
  It contains 6 orederd sets end each set has two generation packets.
- Packet Generator module
  It is responsible to generate gen 3 ordered sets.

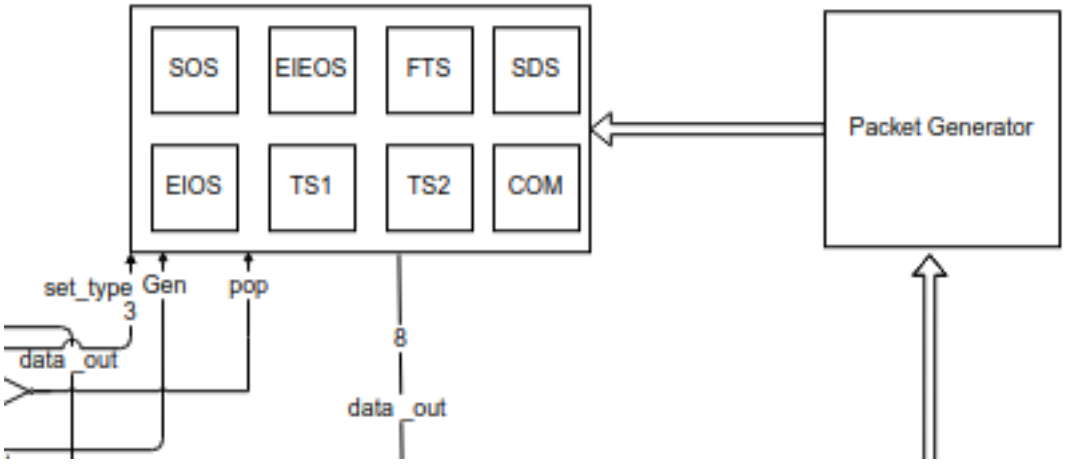

Figure 4.5: Ordered set and packet Generator module

Table 4.1: Input signals for ordered set module

Name	Description
set_type	choose ordered set type (SOS, EIEOS, ...)
Gen	choose generation of the packet gen 1⟶2 or gen 3⟶5
pop	It pops the packet to the bus (data_out) symbol by symbol until finish

Chapter 5

Scenarios

5.1 PIPE interface scenarios

5.1.1 Electrical Idle

The base spec requires that devices send an Electrical Idle ordered set before Tx+/Tx- goes to the electrical idle state.

- The general rules for entring electrical idle state :
  - Strat sending EIOS with asserting TxDataK at the same time.
  - the MAC must always align the electrical idle ordered set on the parallel interface so the first byte of the ordered set is always on the low-order data lines (TxDataK[7:0]).
  - Assert TxElecIdle just after sending the EIOS and make sure that TxDataValid is asserted whenever TxElecIdle toggles.
- The following timing diagram shows an example of electrical idle entry for gen1,2:

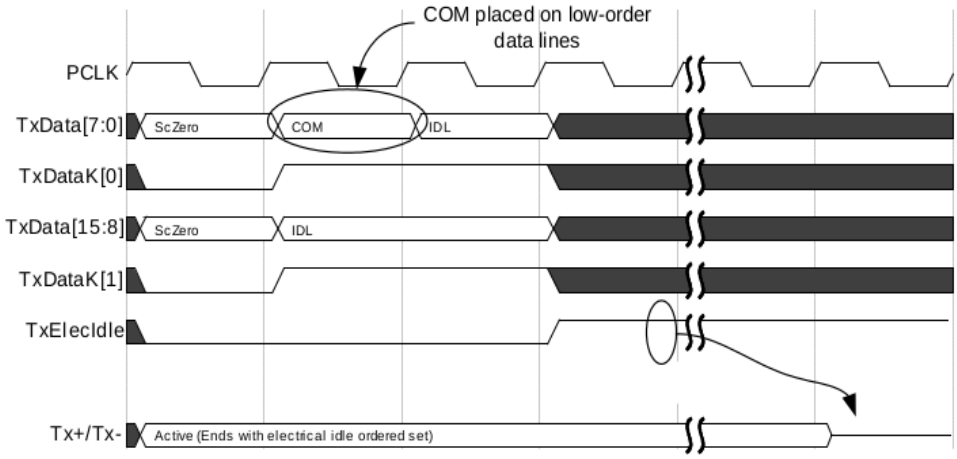

Figure 5.1: electrical idle entry for gen1,2

- Some notes on the previous example :
  - COM appears on the low-order data lines (TxDataK[7:0]).
  - TxElecIdle is asserted after finishing the EIOS.
  - TX+/TX- leaves Active state with the end of EIOS.
- The general rules for leaving electrical idle state :
  - De-assert TxElecIdle and start sending data.
  - Make sure TxDataValid is asserted to ensure sampling TxElecIdle.
- The following timing diagram shows an example of electrical idle entry for gen 3:

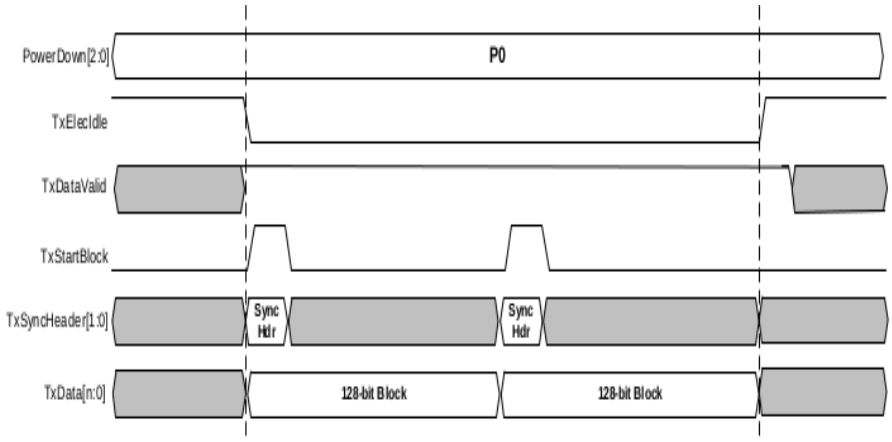

Figure 5.2: electrical idle entry for gen3

Some notes on the previous example:

- TxDataValid can assert earlier before TxElecIdle toggles.
- TxDataValid can de-assert anytime after TxElecIdle asserts as long as it does not overlap with the next Electrical Idle exit sequence.
- TxElecIdle must de-assert at the same clock TxStartBlock asserts.

5.1.2 Link Equalization Evaluation

- While in the P0 power state, the PHY can be instructed to perform evaluation of the current TX equalization settings of the link partner.
- Basic operation of the equalization evaluation is that:
  - the MAC requests the PHY to evaluate the current equalization settings by asserting RxEqEval.
  - When the PHY has completed evaluating the current equalization settings, it asserts PhyStatus for one clock
  - The PHY drives the LinkEvaluationFeedback signals to the appropriate feedback response.
  - The MAC must deassert RxEqEval before initiating another evaluation.
  - To abort an evaluation the MAC de-asserts RxEqEval before the PHY has signaled completion. If the MAC aborts the evaluation the PHY must signal completion as quickly as possible. The MAC ignores returned evaluation values in an abort scenario.

- If a race condition occurs where the MAC aborts by deasserting RxEqEval on same cycle as the PHY asserts PhyStatus then the PHY shall not take any further action.

- The next figure shows an example of the timings for a successful link equalization evaluation request:

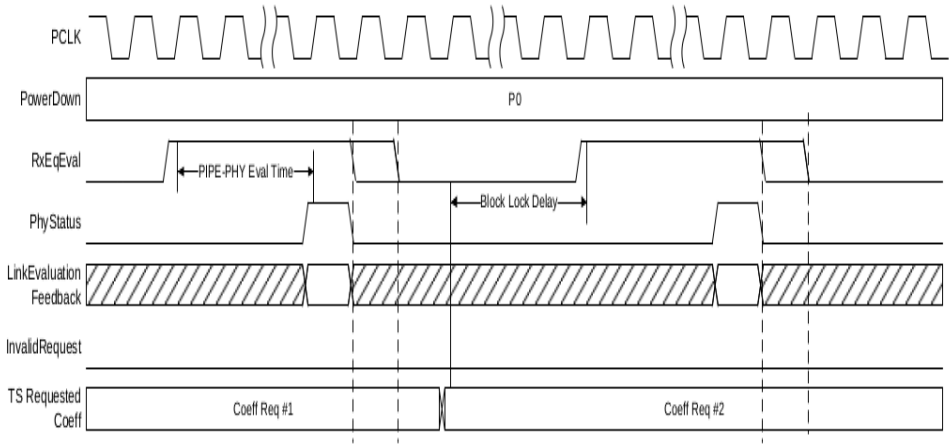

Figure 5.3: successful link equalization evaluation request

- Notes :
  - RxEqEval can de-assert at the same clock the corresponding PhyStatus de-asserts or later as long as RxEqEval de-asserts prior to the next RX Equalization Request.
  - Back-to-back RxEqEval request can happen as close as one clock apart (i.e. RxEqEval can de-assert for one clock before it re-asserts again to start the next RX Equalization request.
  - Once the MAC has requested link equalization evaluation (by asserting RxEqEval), the MAC must leave RxEqEval asserted until after the PHY has signaled completion by the assertion of PhyStatus unless the MAC needs to abort the evaluation due to high level timeouts or error conditions.

- The next figure shows PCI Express 3.0 Equalization Evaluation Request Resulting in Invalid Feedback

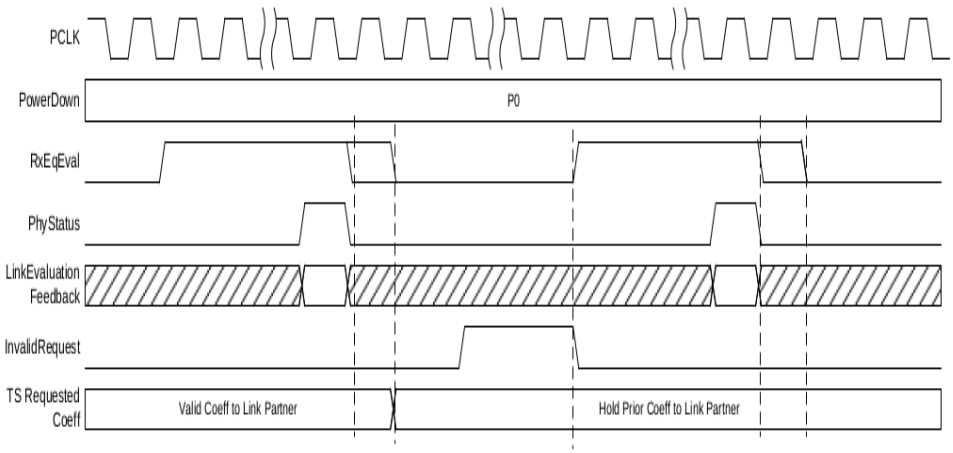

Figure 5.4: Equalization Evaluation RequestResulting in Invalid Feedback

5.1.3 Active PM L0 to L0s and back to L0

- When the MAC and higher levels have determined that the link should transition to L0s, the MAC transmits an electrical idle ordered set and then has the PHY transmitter go idle and enter P0s.
- The general rules for entering L0s state from L0:
  - MAC sends EIOS with the same rules mentioned in the Electrical Idle section.
  - Just after sending the EIOS, the MAC changes PowerDown[1:0] to 01b indicating transitioning to L0s.

  The next figure shows the L0s entry for PCIe gen1/2:

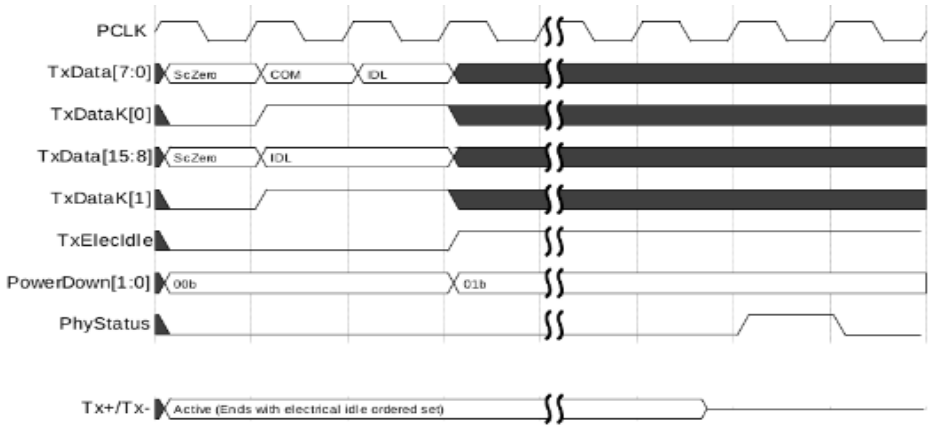

Figure 5.5: L0s entry for PCIe gen1/2

- The general rules for exiting L0s state to L0:
  - The MAC transitions the PHY from the P0s state to the P0 state by changing PowerDown[1:0] to 00b.
  - The MAC waits for the PHY to indicate that it is ready to transmit (by the assertion of PhyStatus).
  - Then the MAC begins transmitting Fast Training Sequences (FTS).
  - The next figure shows an example of L0s to L0 transition when the PHY is running at 2.5GT/s.

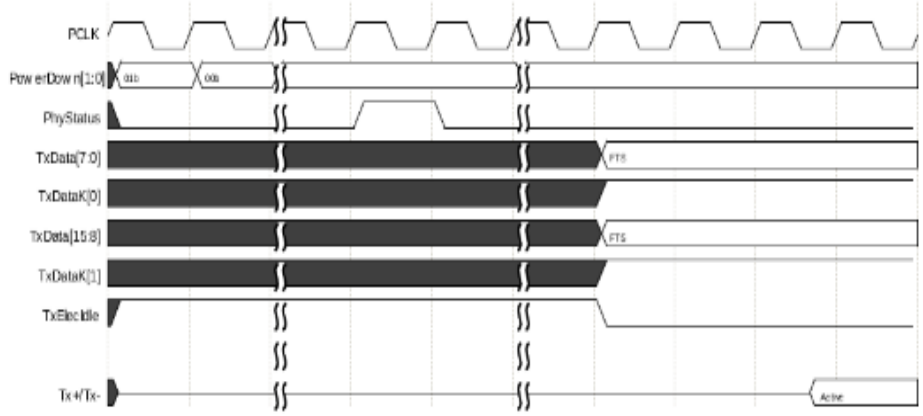

Figure 5.6: L0s to L0 transition

5.1.4 reestablish block alignment in recovery state

- L0 state –> Recovery state due to loss of alignment is detected
- In Recovery.RcrLck/cfg, Mac layer assert commandBlockAlignControl bit using messagebus:
  - Send write committed message to change PHY RX Control4 register to h01 on M2P_Messagebus.
  - The phy send ack back for this message on P2M_Messagebus
- PHY while (commandBlockAlignControl asserted) search for EIEOS to achieve block alignment
- After Blockalignment is finished
- Mac layer go to status L0 and deassert commandBlockAlignControl bit using messagebus:
  - Send write committed message to change PHY RX Control4 register to h00 on M2P_Messagebus
  - The phy send ack back for this message on P2M_Messagebus

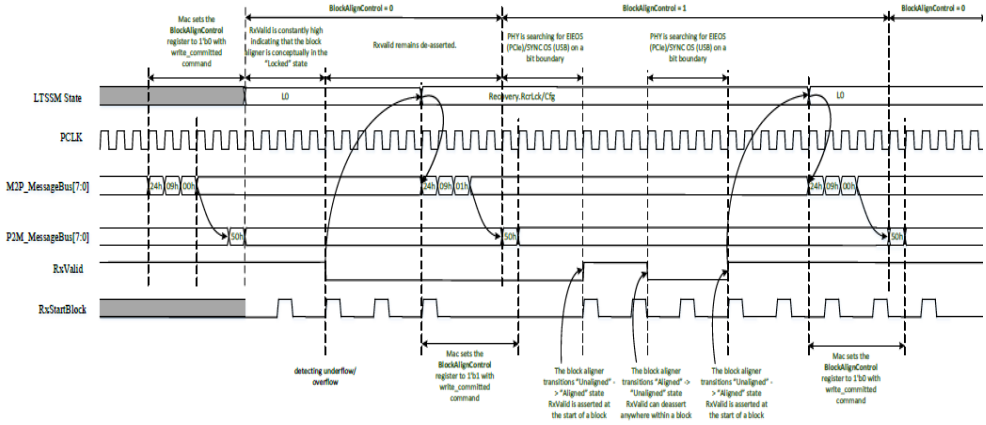

Figure 5.7: reestablish block alignment in recovery state

5.1.5 RXeqTraining

- IN Recovery.Equalization Substate,Mac start EqTraining by asserting RX-EqTraining bit using messagebus:
  - Send write committed message to change PHY RX Control1 register[0] to 1 on M2P_Messagebus

  - The phy send ack back for this message on P2M_Messagebus

- When PHY complete Equalization Training
- PHY layer indicate the status of completion to MAC layer via RxEqualizationDone bit using messagebus:
  - Send write committed message to change RX status register to 1 on P2M_Messagebus
  - The Mac send ack back for this message on M2P_Messagebus

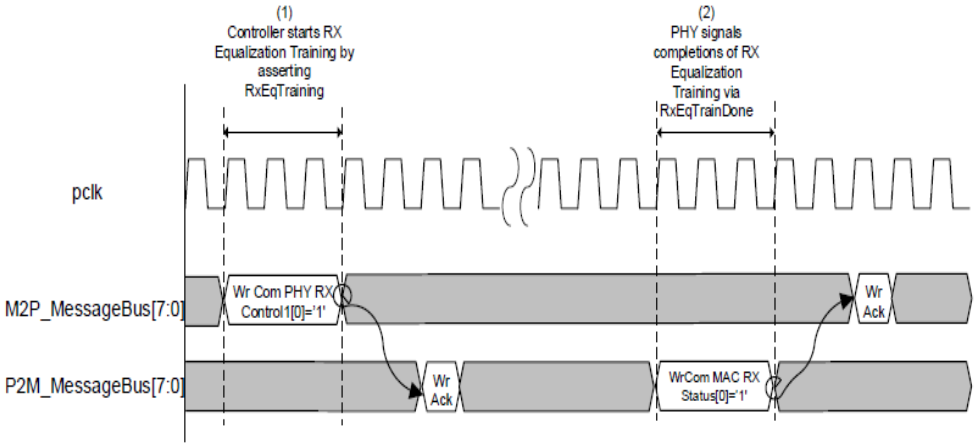

Figure 5.8: RXeqTraining

5.1.6 Reset

- The MAC reset the PHY by asserting Reset signal (active low).
- While Reset is asserted the MAC should have TxDetectRx/Loopback deasserted, TxElecIdle asserted, TxCompliance deasserted and the PowerDown = P1.
- For initial power on the MAC must holds PHY in reset until the power and CLK to the PHY are stable.
- The PhyStatus is signal is high during reset until reset is deasserted and PCLK is running in operational frequency for one cock and the PHY is in specified power state then it will be deasserted.
- When PhyStatus deasserted after the deassertion of the Reset, The MAC can perform any operational sequences.

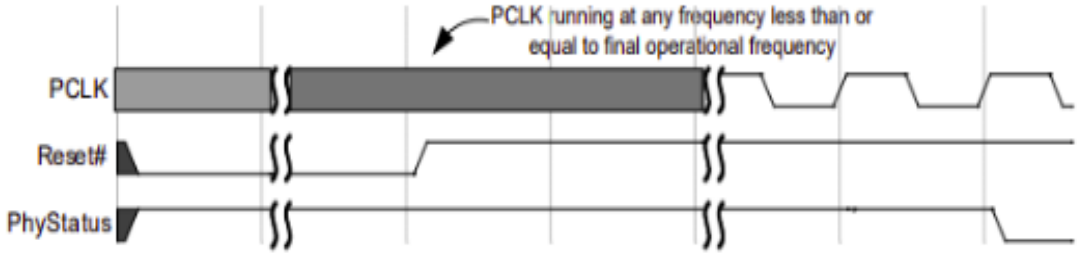

Figure 5.9: Reset

5.1.7 Updating TxDeemph

The TXDeemph is 18-bit variable divided into three parts on (PHY TX Control2, PHY TX Control3, PHY TX Control4) registers.

- The MAC sends on M2P_MessageBus write_committed to TxControl5 register to make GetLocalPresetCoefficients request for one or more values of LocalPresetIndex to update the TxDeemph value. This field is used to make dynamic preset coefficient updates.
- The PHY acknowledge the request by sending write_ack on the P2M_MessageBus.
- The PHY sends three messages on the P2M_MessageBus to update the LocalPresetCoefficients fields in the (Tx Status0, Tx Status1, Tx Status2) registers.
- The MAC sends write_ack message on M2P_MessageBus to ensure that the LocalPresetCoefficients fields are updated successfully.
- The MAC sends three messages on the M2P_MessageBus to the PHY.
  - Two of them are write_ucommitted.
  - One write_committed to ensure that data are written in the (PHY TX Control2, PHY TX Control3, PHY TX Control4) registers.
  - for TxDeemph, the new TxDeemph value must be reflected on the pins within 128ns.
- After the write_committed for TxDeemph, the new TxDeemph value must be reflected on the pins within 128ns.

Note: For every GetLocalPresetCoefficients request, there is a 128ns maximum response time for the PHY to return the LocalTxPresetCoefficients value.

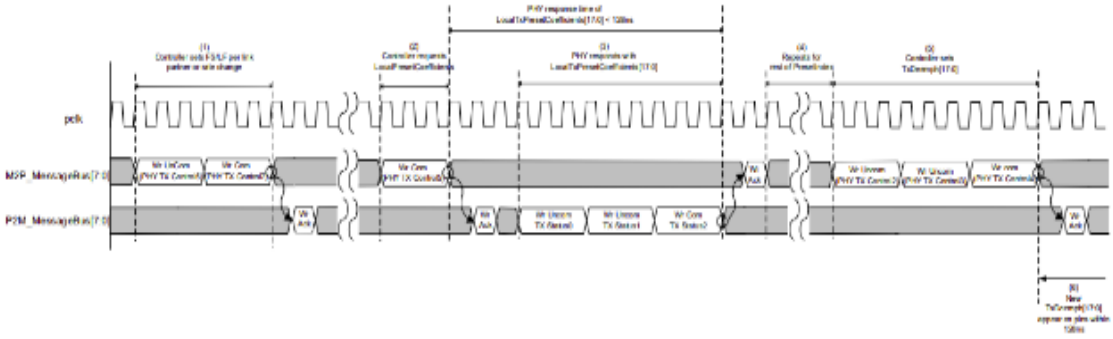

Figure 5.10: Updating TxDeemph

5.1.8 Selecting De-emphasis

- While in the P0 power state and transmitting at 5.0GT/s, 8.0 GT/s, 16 GT/s or 32 GT/s, the PHY can be instructed to change the value of the transmitter equalization.
- Steps of operation:
  - The MAC changes the TXDeemph value with the new Tx-De-emphasis value.
  - If the signaling rate is 5GT/S, the PHY must be capable of transmitting with the new setting within 128 ns.
  - If the signaling rate is 8.0 GT/s, 16 GT/s, or 32 GT/s, the PHY must be capable of transmitting with the new setting within 256 ns.

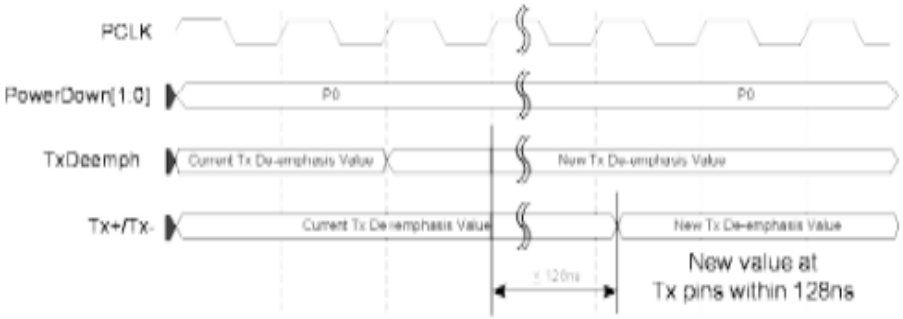

Figure 5.11: Selecting Tx De-emphasis value

Polarity Inversion

- RxPolarity signal must be asserted to make polarity inversion.
- The PHY must invert received data after RxPolarity is asserted.
- Inverted data must be showing up on RxData within 20 PCLKs after the assertion of RxPolarity.
- Note: The D21.5 symbol when inverted it will be D10.2 symbol as shown in figure.

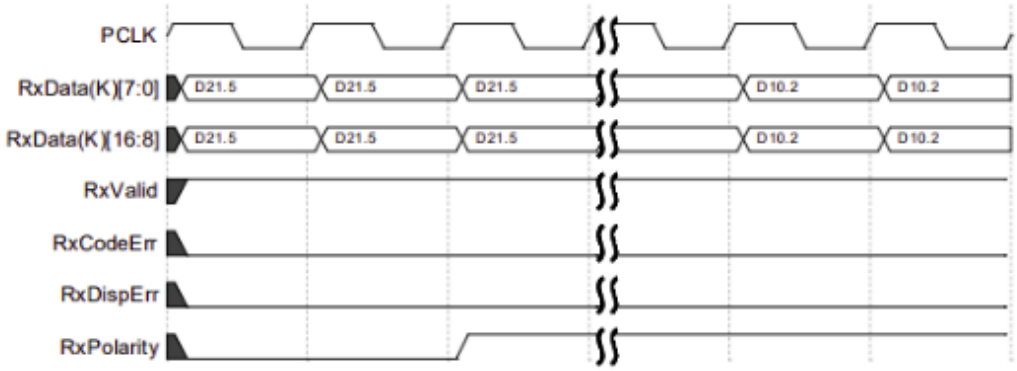

Figure 5.12: Polarity inversion

5.1.9 Signal rate

- Signal rate can be change when PHY in L0 or L1 power state and Tx Electrical IDL is asserted.
- After the MAC changes PCLK rate, the change to the PCLK can happen only after the PclkChangeOk has been driven high by the PHY.
- The MAC changes the input PCLK and then handshakes by asserting PclkChangeAck.
- The PHY responds by the asserting PhyStatus for one input PCLK cycle and deasserts PclkChangeOk on the trailing edge of PhyStatus.
- The MAC deasserts PclkChangeAck when PclkChangeOk is sampled low and may deassert Tx Electrical IDL.

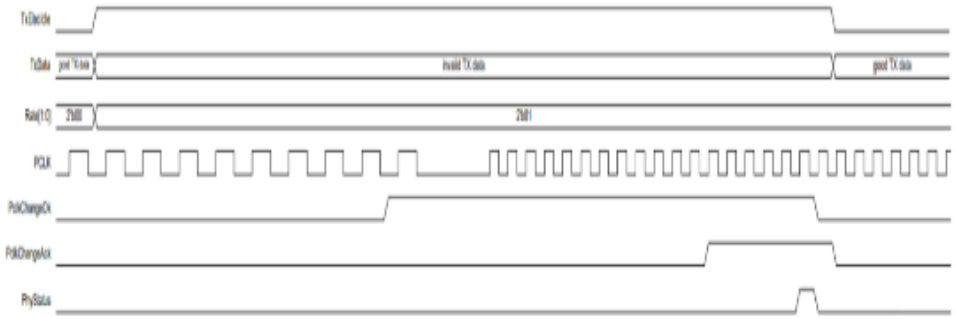

Figure 5.13: signal rate

5.1.10 Error Detection

- The PHY is responsible for detecting receive errors of several types, these errors are signalled to the MAC layer using the receiver status signals (RxStatus[2:0]).
- When a receive error occurs, the appropriate error code is asserted for one clock cycle at the point in the data stream across the parallel interface closest to where the error actually occurred.
- There are four error conditions that can be encoded on the RxStatus signals.
- If more than one error should happen to occur on a receive byte, the errors should be signaled with the priority shown below.
  - 8B/10B decode error or block code error (RxStatus=100b).
  - Elastic Buffer overflow (RxStatus=101b).
  - Elastic Buffer underflow (RxStatus=110b).
  - Disparity errors (RxStatus=111b).

5.1.11 8B/10B Decode Errors

- For a detected 8B/10B decode error, the PHY should place an EDB symbol in the data stream in place of the bad byte, and encode RxStatus with a decode error during the clock cycle when the effected byte is transferred across the parallel interface.
- For greater than 8-bit interface, if the bad on the lower byte lane, one of the other bytes may have bad disparity, but the 8B/10B error has the precedence.

- In the example below, the receiver is receiving a stream of bytes Rx-a through Rx-z, and byte Rx-f has an 8B/10B decode error, in the place of that byte, the PHY places an EDB on the parallel interface and sets RxStatus to the 8B/10B decode error code.

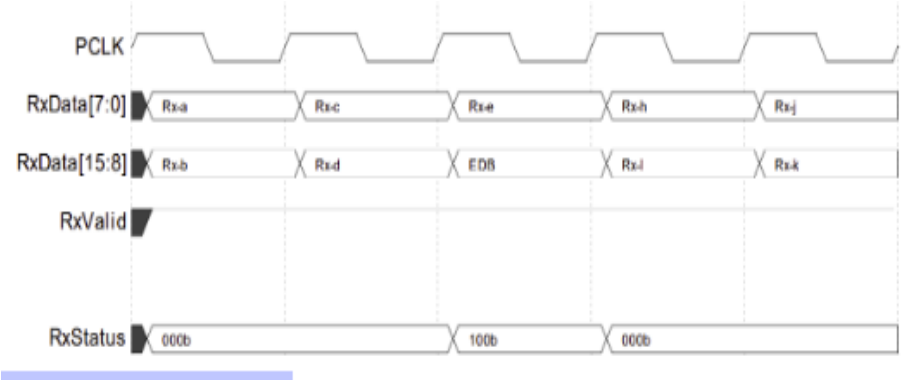

Figure 5.14: 8B/10B Decode Errors

5.1.12 Disparity Errors

- For a detected disparity error the PHY should assert RxStatus with the disparity error code during the clock cycle when the affected byte is transferred across the parallel interface.
- For greater than 8-bit interfaces, it is not possible to discern which byte (or possibly both) had the disparity error.
- In the example below, the receiver detected a disparity error on either (or both) Rx-e or Rx-f data bytes and indicates this with the assertion of RxStatus.

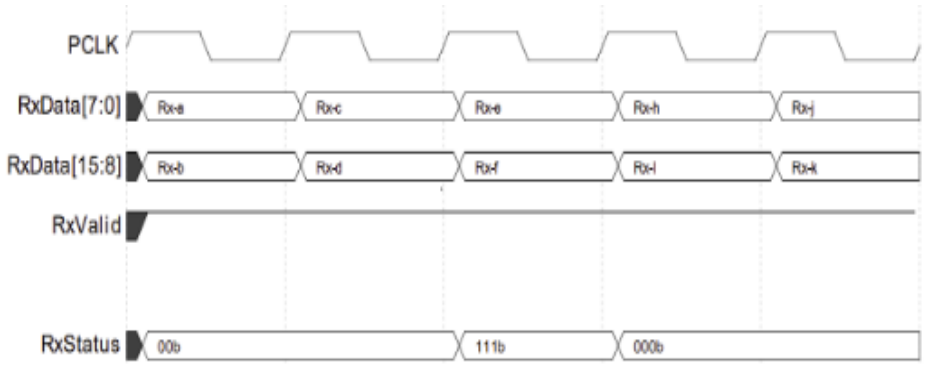

Figure 5.15: Disparity Errors

- Note: MACs often treat 8B/10B errors and disparity errors identically.

5.1.13 Elastic Buffer Errors

Elastic Buffer Underflow

- For elastic buffer errors, an underflow should be signaled during the clock cycle or clock cycles when a spurious symbol is moved across the parallel interface, the symbol moved across the interface should be the EDB symbol.
- In the timing diagram below, the PHY is receiving a repeating set of symbols Rx-a thru Rx-z, the elastic buffer underflows causing the EDB symbol to be inserted between Rx-g and Rx-h symbols, the PHY drives RxStatus to indicate buffer underflow during the clock cycle when the EDB is presented on the parallel interface.

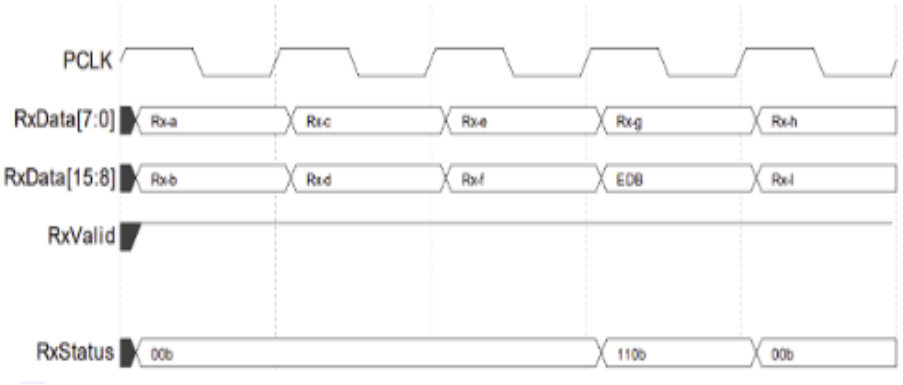

Figure 5.16: Elastic Buffer Underflow

Elastic Buffer Overflow

- For an elastic buffer overflow, the overflow should be signaled during the clock cycle where the dropped symbol or symbols would have appered in the data stream.
- For the 16-bit interface it is not possible, or necessary, for the MAC to determine exactly where in the data stream the symbol was dropped.
- In the timing diagram below, the PHY is receiving a repeating set of symbols Rx-a thru Rx-z, the elastic buffer overflows causing the symbol Rx-g to be discarded, The PHY drives RxStatus to indicate buffer overflow during clock cycle when Rx-g would have appeared on the parallel interface.

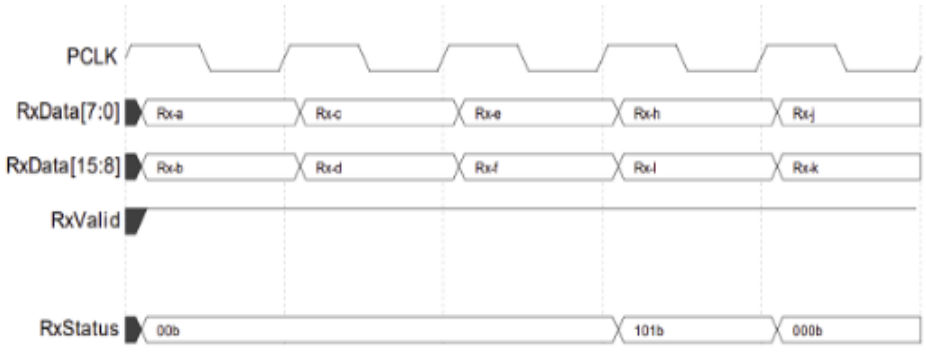

Figure 5.17: Elastic Buffer Overflow

5.1.14 Receiver detection

- While in the P1 power state the PHY can be instructed to perform a receiver detection operation to determine if there is a receiver at the other end of the link.
- Basic operation of receiver detection is that MAC requests the PHY to do receiver detect sequence by asserting TxDetectRx/Loopback.
- When the PHY has completed the receiver detect sequence, it asserts PhyStatus for one clock and drives the RxStatus signals to the appropriate code.
- After the receiver detection has completed, the MAC must deassert TxDetectRx/Loopback before initiating another receiver detection.
- Once the MAC has requested a receiver detect sequence the MAC must leave TxDetectRx/Loopback asserted until after the PHY has signaled completion by the assertion of Phystatus.

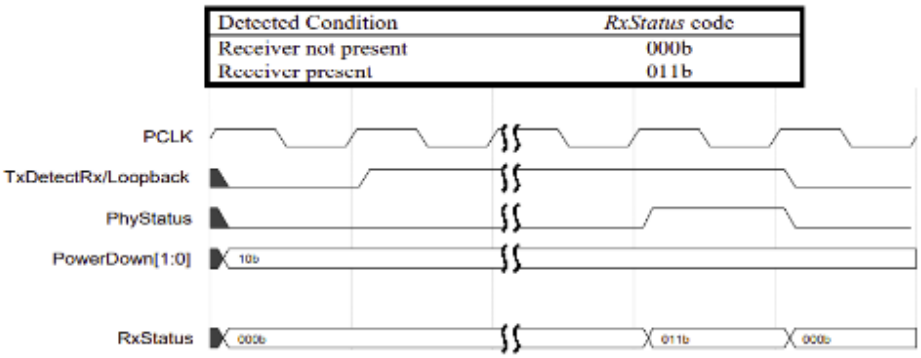

Figure 5.18: Receiver detection

5.1.15 128b/130b Encoding and Block synchronization

- For every block (usually 128 bits) that is moved across the PIPE TxData interface at the 8.0 GT/s rate, 16 GT/s rate, or 32 GT/s rate, TxSyncHeader signal is asserted to Provide the Sync Header for the PHY to use in the next 128b block.
- The MAC must assert the TxDataValid signal to indicate that PHY will use the data.
- TxStartBlock signal is asserted to allow the MAC to tell PHY the starting byte for a 128b block.

- TxElecIdle signal Forces Tx output to electrical idle when asserted.
- For every block (usually 128 bits) that is moved across the PIPE RxData interface at the 8.0 GT/s rate, 16 GT/s rate, or 32 GT/s rate, RxSyncHeader signal is asserted to Provide the Sync Header for the MAC to use in the next 128b block.
- The PHY must assert the RxDataValid signal to indicate that MAC will use the data.
- RxStartBlock signal is asserted to Allows the PHY to tell MAC the starting byte for a 128b block.
- RxElecIdle signal Indicates receiver detection of an electrical idle when asserted.

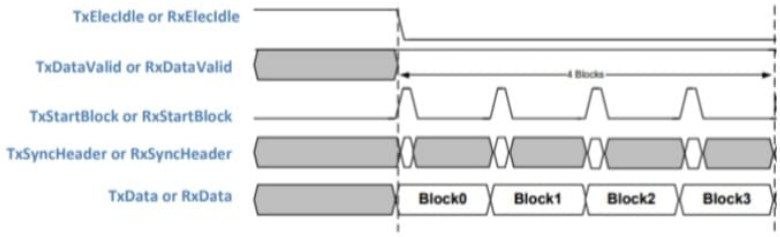

Figure 5.19: 128b/130b Encoding and Block synchronization