PCIe version 5.0 Mac Layer Implementation

Mac Layer
Dec 15, 2023

Contents

Contents

Referenced Documents

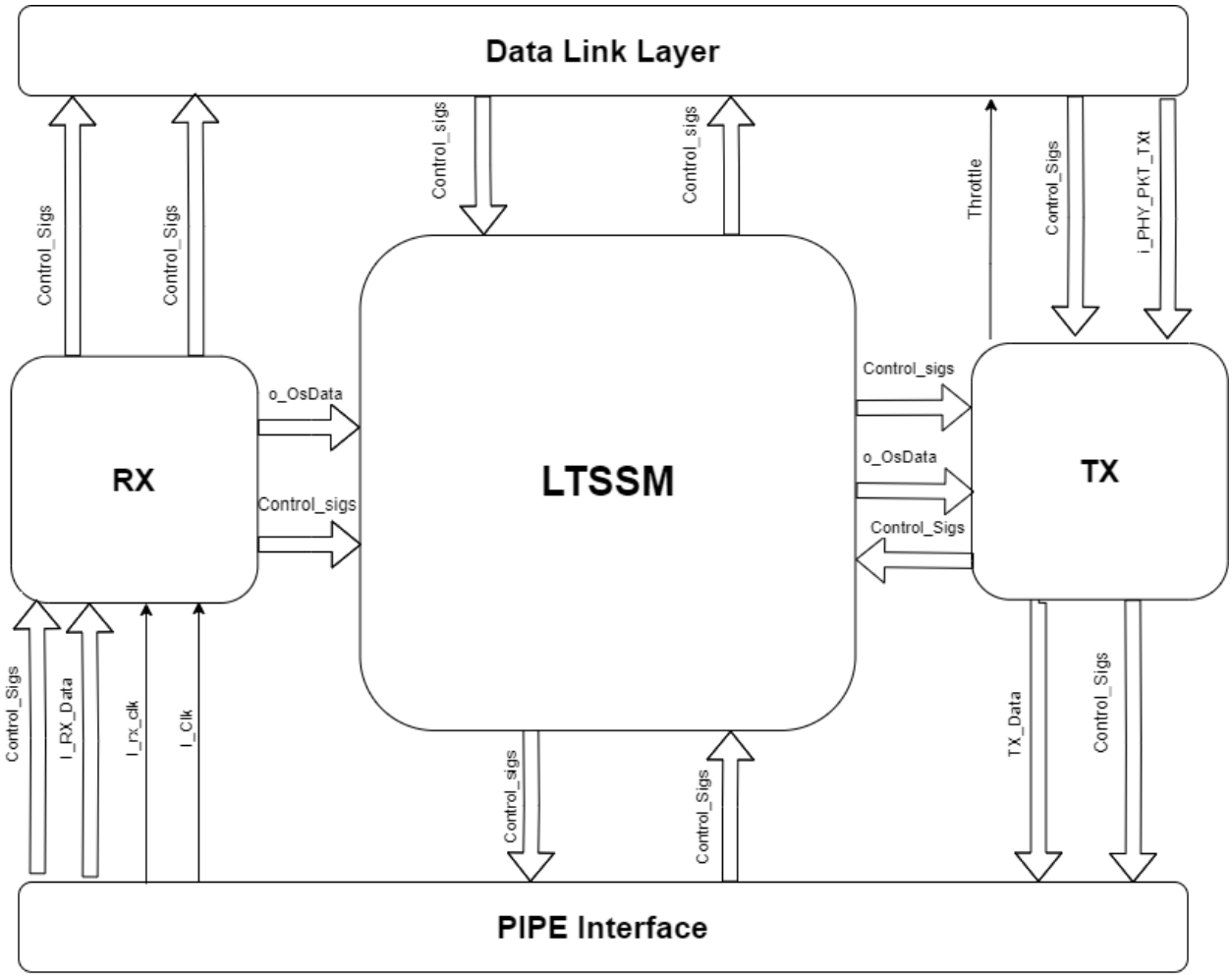
1- MindShare - PCI Express Technology 3.0.

https://www.mindshare.com/Books/Titles/PCI_Express_Technology_3.0

2- PCI Express® Base Specification Revision 5.0 Version 1.0

Overview

❖ High-Level overview:



- Physical layer is the lowest layer in PCIe architecture as TLP and DLLP packets are forwarded down from Data Link Layer for transmission over link and forwarded up to the Data Link Layer at the Receiver so, Physical Layer isolate Transaction and Data Link Layer from signaling technology used for link data interchange.

❖ Features and limitations:

Supported Features	Non-Supported Features
Link Recovery	Lane Reversal Features
Operating at 32 lanes or 1 lane	Intermediate lane configuration
SKP removal	Compliance substate in polling state
Inclusion of 128/130 encoding	Hot Reset, Disable states
Inclusion of elastic buffer	Bad packets
SERDES Interface	Loopback state
Lane-Lane De-skew	Low Power states
DC-Balance	Downstream ports
Receiver error reporting	
Compatibility with GEN 1	
Support Upstream ports	
PIPE Standard interface	

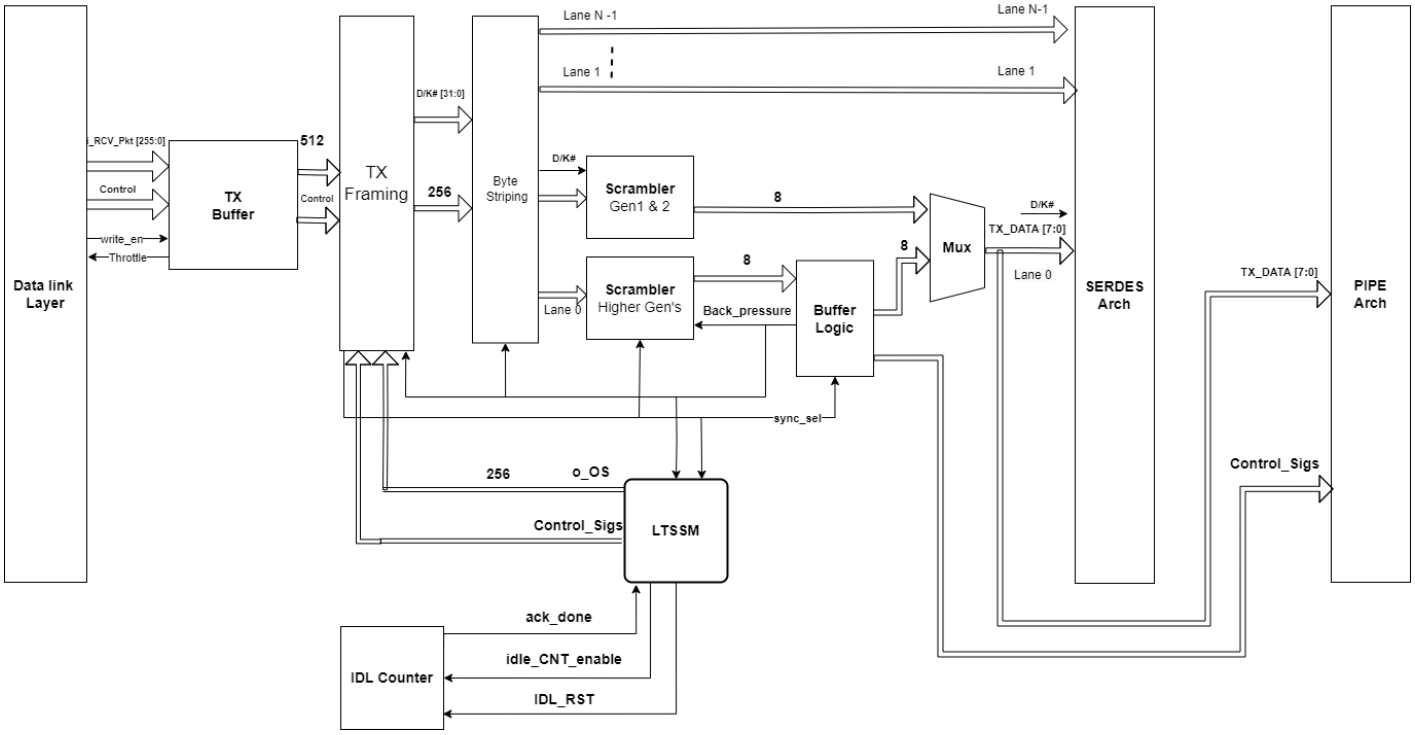
❖ Dependencies and Assumptions

- Fixed data bus with the data link layer of 32 bytes.
- Fixed data bus with 8 bits with PIPE interface in each lane.
- Throttling the data link layer when needed.
- Maximum number of lanes is 32.
- Start sending TLP packet even though the whole packet is not sent from data link layer yet.
- All errors are considered receiver errors.
- Data link layer discards the packet associated with a reported error.
- TX, RX Buffers are flushed when Receiver error.
- PIPE is responsible for implementing Encoding and Decoding blocks in lower GEN.

❖ Functional description

1. Transinmter (TX)

- **TX Architecture**



- **TX Overview**

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

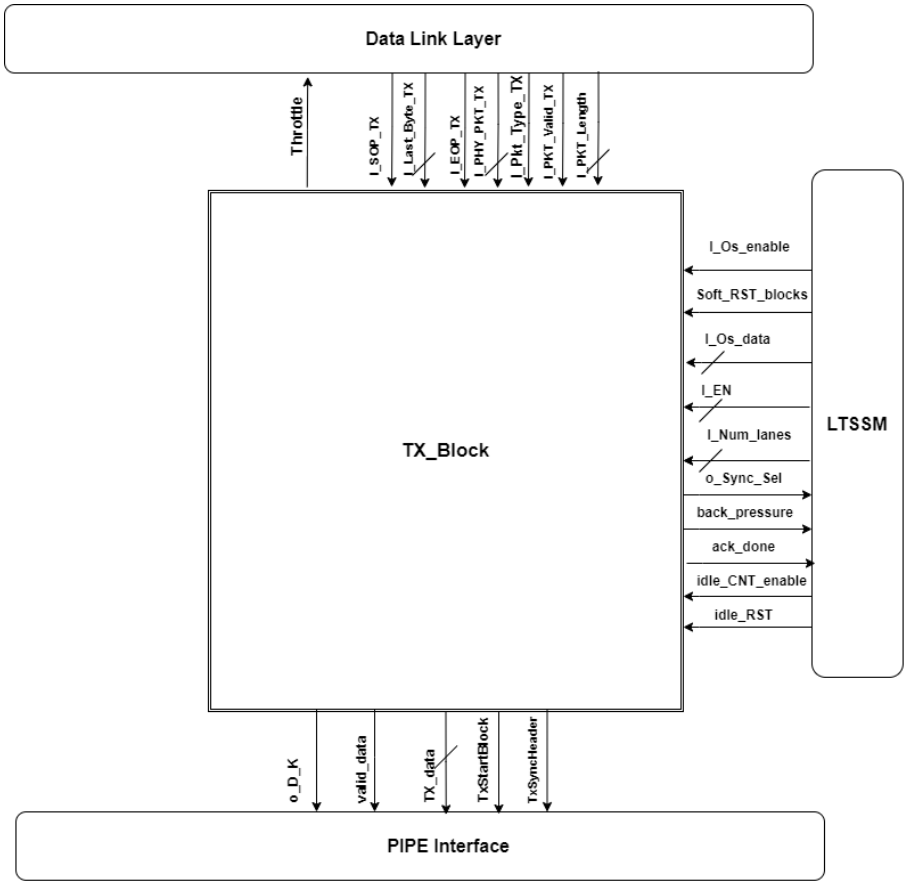
    1. Receive the packets from the data link layer either it is TLP or DLLP.
    2. Capsulate the TLP and DLLP with control characters (STP - SDP - END - EDB).
    3. Send the ordered set packets from the LTSSM if it enabled the Frame control.
    4. Handle the priority between the packets coming from the 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.

6. Organize the Framed data on the supported lanes with certain format.

7. Scrambling the data on each lane to avoid crosstalk noise or electromagnetic interference.

8. Encoding the data on each lane to maintain DC balancing



- **TX Interface with PIPE & LTSSM & Data Link Layer**

A. Interface with Data Link Layer

Name	Direction	Size	Description
I_PHY_PKT_TX	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.
Throttle	Output	1 bit	Output to the data link layer The full buffer indicator
I_PKT_Type_TX	Input	1 bit	Input from the data link layer It determines whether the packet is TLP or DLLP. 1 for TLP and 0 for DLLP.
I_SOP_TX	Input	1 bit	Input from the data link layer The start of packet indicator.
I_Last_Byte_TX	Input	5 bits	Input from the data link layer It determine the Last byte of the packet to insert the END control character.
I_EOP_TX	Input	1 bit	Input from the Data link Layer. The end of packet indicator.
I_PKT_Valid	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_PKT_Length_TX	Input	11 bits	Input from the data link layer It determine the length of the packet as we dispense with end character in higher GEN'S.

B. Interface with LTSSM

Name	Direction	Size	Description
I_Os_data	Input	3 bits	Input from LTSSM It determine the supported number of lanes.
I_Os_enable	Input	1 bit	Input from LTSSM It enable the frame control to send order sets on the link. .
I_Num_Lanes	Input	5 bit	Input from LTSSM It determine the supported number of lanes.
I_EN	Input	1 bit	Input from LTSSM It enable all TX path blocks to operate.
Soft_RST_blocks	Input	1 bit	Input from LTSSM to soft reset TX Buffer in Link retrain due to error handling.
Idle_CNT_enable	Input	1 bit	Input from LTSSM to enable IDL counter to count until 16 counts.
IDL_rst	Input	1 bit	Input from LTSSM to reset IDL counter.
o_Sync_Sel	Output	1 bit	Output from TX to enable OS_creator to send OS's.
back_pressure	Output	1 bit	Output from TX to enable OS_creator to send OS's.
ack_done	Output	1 bit	Output from TX to ack LTSSM that we have completed already 16 counts.

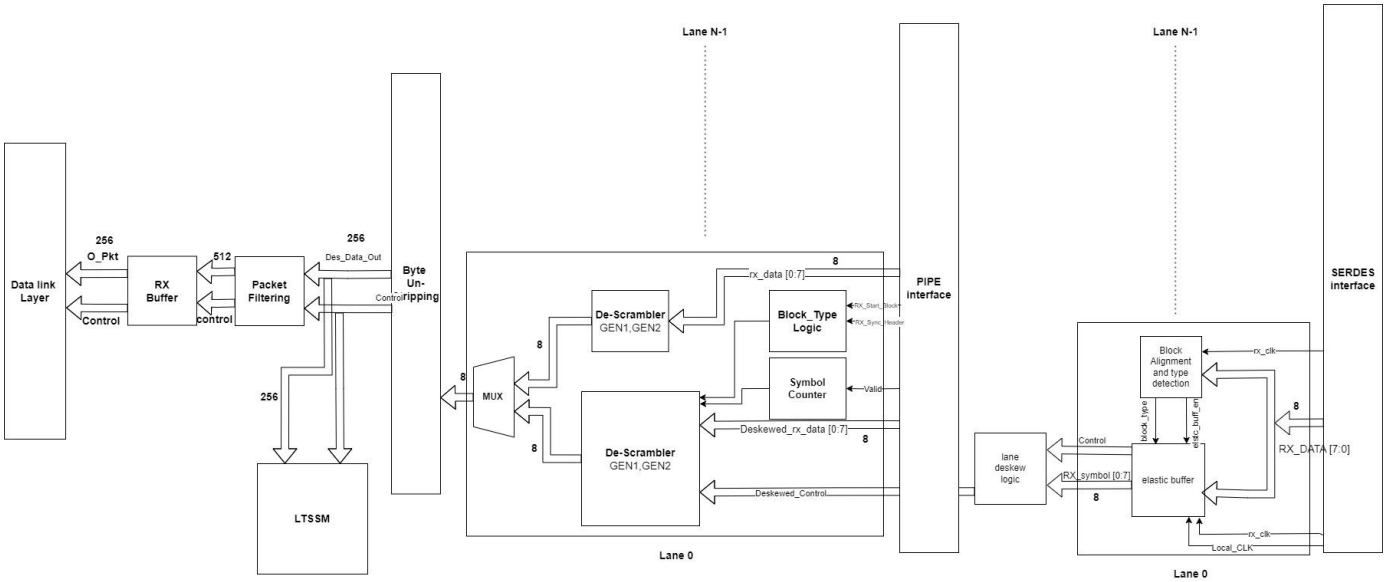
C. Interface with PIPE

Name	Direction	Size	Description
TX_DATA	Output	8 bits	Output to PIPE Data transmitted on the link.
valid_data	Output	1 bit	Output to PIPE to valid data transmitted to PIPE to enable PIPE

			send accumulated location in case of back pressure in PIPE Standard interface.
o_D_K	Output	1 bit	Output to PIPE to indicate the Data transmitted that it's control characters or data in LOW GEN.
TxSyncHeader	Output	2 bits	Output to PIPE to indicate the transmitted data that it is OS's or Data in HIGH GEN in PIPE Standard interface.
TxStartBlock	Output	1 bit	Output to PIPE to indicate the Start of the block in PIPE Standard interface.

2. Reciver (RX)

- **RX Architecture**

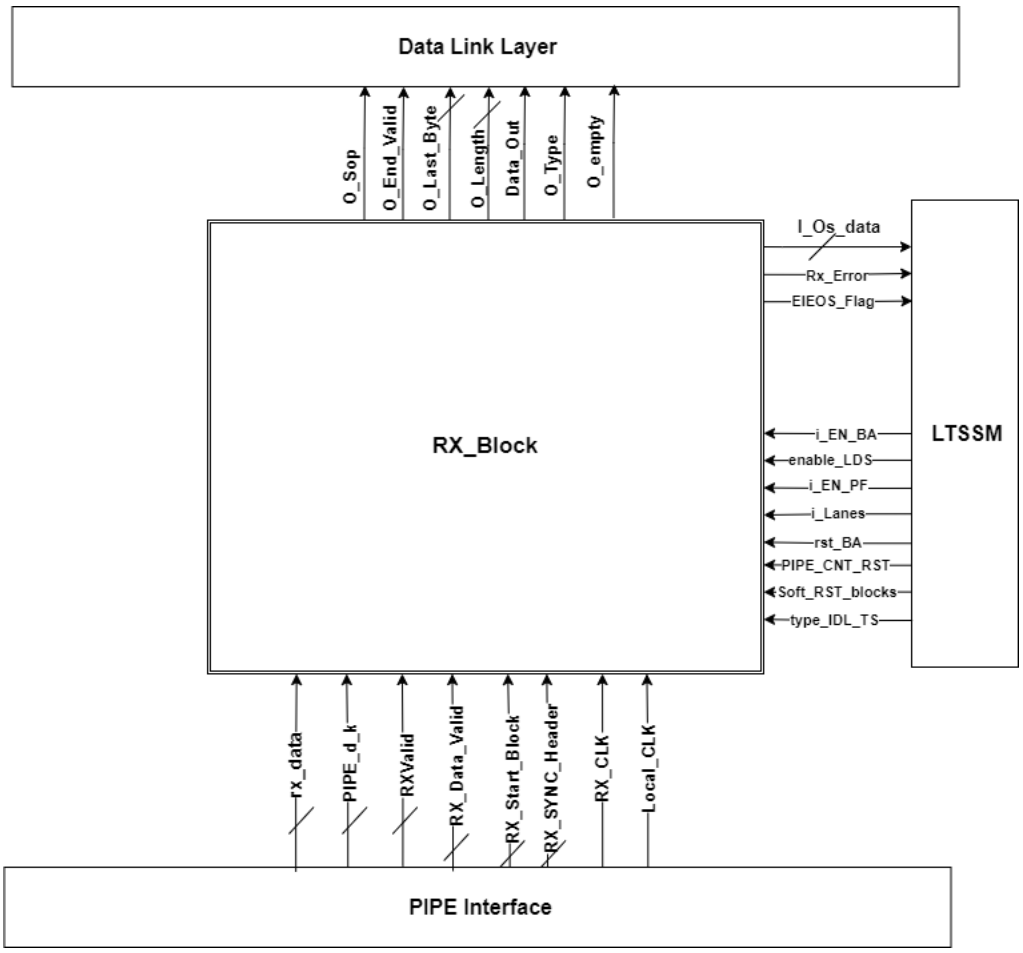


- **RX Overview**

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

    1. Receiving the packets and bypassing them to the data link layer either it is TLP or DLLP.
    2. Filter the TLP and DLLP from framing (STP - SDP).
    3. Bypass the ordered set packets to the LTSSM.
    4. Descrambling the data on each lane to avoid crosstalk noise or electromagnetic interference.
    5. Performing the block alignment process to determine the start of a block and remove the sync header.
    6. Removing SKP symbols, when the difference between the local clock and the recovered clock exceeds a certain threshold to avoid buffer overflow.
    7. Achieving the process of the lane-to-lane deskew, so that all lanes are aligned.
    8. Reporting the detection of an error to the LTSSM to initiate the recovery process.

- **RX Interface with PIPE & LTSSM & Data Link Layer**



A. Interface with Data Link Layer

Name	Direction	Size	Description
Data_Out	Output	256 bits	**Output to the data link layer** The data which is sent to the data link layer. If the packet is more than 32 Bytes, each 32 bytes of it are sent in a single clock cycle.
O_Type	Output	1 bit	**Output to the data link layer** It determines whether the packet is TLP or DLLP. 1 for TLP and 0 for DLLP.

O_Sop	Output	1 bit	**Output to the data link layer** The start of packet indicator.
O_Last_Byte	Output	5 bits	**Output to the data link layer** It indicates how many bytes of the Data_Out are valid data locations.
O_End_Valid	Output	1 bit	**Output to the data link layer** It determine whether the Data_Out contains the end of a packet or not.
O_Length	Output	11 bits	**Output to the data link laye** It determine the length of the packet as we dispense with end character in higher GEN'S.
O_Empty	Output	1 bit	**Output to the data link layer** It tells the data link layer that the buffer is empty.

B. Interface with LTSSM.

Name	Direction	Size	Description
type_IDL_TS	Input	1 bit	**Input from the LTSSM** Asserted when an OS is received in L0, to set the PIPE counter to 1 to decode the proceeding OSs correctly.
Soft_RST_blocks	Input	1 bit	**Input from the LTSSM** Asserted in the recovery process to reset the states of all RX blocks, and clear all the buffers
PIPE_CNT_RST	Input	1 bit	**Input from the LTSSM** Asserted to Reset the PIPE Counter in specific LTSSM transitions.
rst_BA	Input	1 bit	**Input from the LTSSM** Asserted when the LTSSM transitions from L0 to the recovery state, used to clear the error of the Block Alignment block.
i_Lanes	Input	1 bit	**Input from the LTSSM**

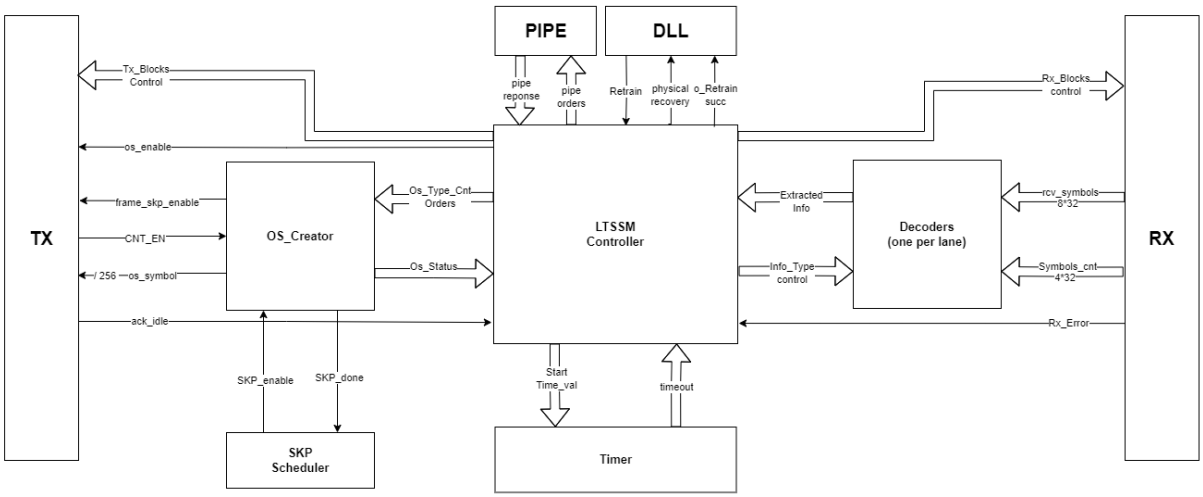
			Indicates the number of configured lanes during the link training process. 1'b1: 32 lanes 1'b0: 1 lane
i_EN_PF	Input	1 bit	**Input from the LTSSM** Asserted when the speed change process is completed successfully, and the transition is made again to L0 to enable the processing of the packet filter.
enable_LDS	Input	1 bit	**Input from the LTSSM** Asserted when the speed change process is completed successfully to enable the process of de-skewing the lanes.
i_EN_BA	Input	1 bit	**Input from the LTSSM** Asserted when the speed change process is completed successfully to enable the process Block alignment.
Des_Data_Out	Output	256 bits	**Output to the LTSSM** Descrambled Data passed to the LTSSM.
RX_Error	Output	1 bit	**Output to the LTSSM** Asserted when an error is detected from the packet filter, lane to lane deskew or the block alignment (e.g: OS without EDS before it, Undefined Sync header)
EIEOS_Flag	Output	1 bit	**Output to the LTSSM** Asserted when the Packet Filter receives an EIEOS in L0, indicating that the other has entered recovery, so the LTSSM must transition to recovery as well.

C. Interface with PIPE.

Name	Direction	Size	Description
rx_data	Input	256 bits	**Input from PIPE** Data received from the link.
PIPE_d_k	Input	32 bits	**Input from PIPE** Used in lower GENs only, with PIPE Interface Indicates whether the symbol is data or control character.
RXValid	Input	32 bits	**Input from PIPE** Used in lower GENs only, with PIPE Interface Indicates Symbol lock and valid data on rx_data bus.
RX_Data_Valid	Input	32 bits	**Input from PIPE** Used in Higher GENs only, with PIPE Interface as it enables the PHY to direct the MAC to disregard the data interface for one clock cycle, due to the sync header removal, When set to one, it indicates that the PHY will utilize the data, and when set to zero, it indicates that the PHY will not utilize the data.
RX_Start_Block	Input	32 bits	**Input from PIPE** Used in Higher GENs only, with PIPE Interface, it indicates the start of a new 128/130 block.
RX_SYNC_Header	Input	2*32 bits	**Input from PIPE** Used in Higher GENs only, with PIPE Interface, hold the value of the sync header on the lane, to differentiate whether the upcoming block is data or OS, the signal is meaningful only at the assertion of RX_Start_Block
RX_CLK	Input	1 bit	**Input from PIPE** RX recovered CLK.
Local_CLK	Input	1 bit	**Input from PIPE** RX local CLK.

3. Link Training Status State Mashine (LTSSM)

- ## LTSSM Architecture



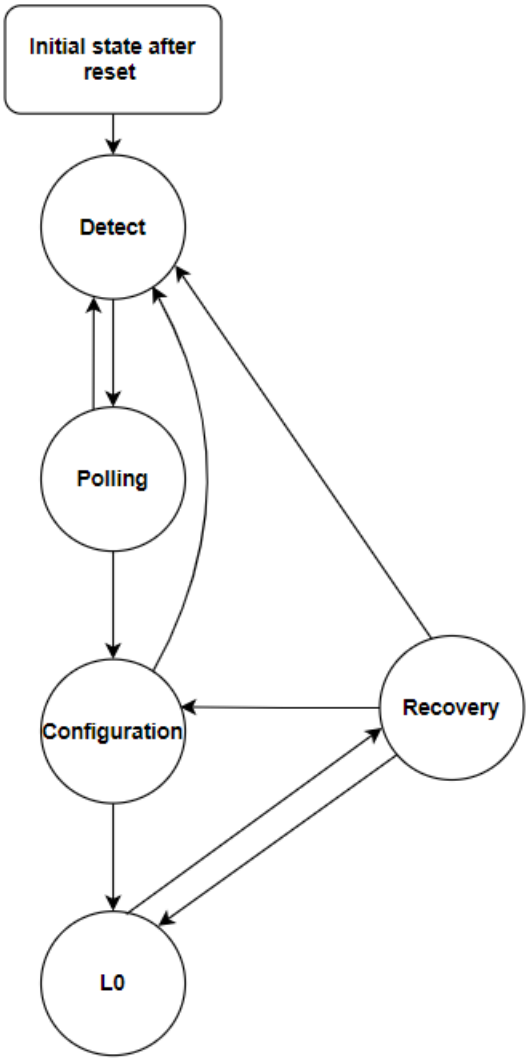
- ## LTSSM Overview

- The functions of the LTSSM block can be summarized in the following point:
  1. Initialization process of the link.
  2. Training process to make the link ready for Transmission and receiving.
  3. Retraining Process when instructed from data link layer or software.

- These training processes are initiated automatically by the hardware and managed by the LTSSM to achieve multiple things such as:
  1. Bit lock: where the clock and data recovery logic recover the transmitter clock from the incoming bit stream.
  2. Symbol Lock: In 8b/10b encoding, after the bit lock, the receiver can now see the bits coming, but still need to find the boundaries of the symbol (10 bits), as this is the unit the receiver will deal with.
  3. Block Lock: In 128/130b encoding, the receiver needs to find the boundaries of each block, to process the incoming symbols within it.
  4. Link Width: the two connected devices need to negotiate at the link width that they are going to use.

5. Polarity Inversion: The D+ and D- differential terminals of the two connected devices may be reversed, hence this must be detected and solved during link training.
6. Link Data Rate: The two connected devices may support different data rates, as a result these rates must be advertised during link training, to settle on the highest possible rate for both, that makes the link works without any reliability problems.
7. Lane to Lane De-skew: As the connected devices have a parallel connection consisting of multiple lanes, the data may arrive at different times at each lane, and this must be resolved during the training.

- **LTSSM States**



- The LTSSM consists of 11 high level states but the scoope of this design is on only 5 of them:
    1. Detect: The initial state, where the device checks whether there is a receiver at the end of the link or not.
    2. Polling: In this state, transmitters and receivers exchange TS1s and TS2s to achieve bit lock, symbol lock(8b/10b), block alignment(128b/130b), resolve polarity inversion and advertise the supported data rates.
    3. Configuration: In this state, transmitters and receivers exchange TS1s and TS2s to determine: the link width, assign lane numbers, and de-skew lane to lane timing differences.
    4. L0: This is the normal, operational state, where TLPs and DLLPs are exchanged between the two connected devices.
    5. Recovery: This state is entered when the link needs retraining, if an error has occurred while in L0, or to recover from low power states, or to change the link speed to a higher rate than 2.5GT/S if a higher speed is supported, or to change link width due to reliability or power management reasons.

• LTSSM Interface with PIPE & Data Link Layer

A. Interface with Data Link Layer.

Name	Direction	Size	Description
O_Link_Up	Output	1 bit	Informs the data link layer that the training of the link has finished, and the link is now operational, ready to receive data.
O_Retrain_succ	Output	1 bit	Informs the data link layer the retraining has finished successfully.
O_Actual_pwr	Output	2 bits	Informs the Data Link Layer with the power state. 00: L0 01: L0s 10:L1 11:L2
O_enable	Output	1 bit	Asserted when a change has been made to the power state.

I_Pwr_States	Input	2 bits	The data link layer direct the physical layer to a certain power state. 00: L0 01: L0s 10:L1 11:L2
I_enable	Input	1 bit	Asserted by the Data link layer when the value of I_Pwr_states changes.
Retrain	Input	1 bits	Asserted by the data link layer to instruct the physical layer into recovery to retrain the link.
Physical_recovery	output	1 bit	Asserted when a receiver error is detected or an EIEOS is received, which indicates MAC layer has entered recovery.

B. Interface with PIPE.

Name	Direction	Size	Description
O_St_Detect	Output	1 bit	Directs the PIPE to start receiver detection.
O_E_Idle	Output	1 bit	Forces TX output to electrical Idle
O_Polarity_inv	Output	1 bit	Asks the PIPE to perform polarity inversion on the received data
O_Pwr_States	Output	2 bits	Power the transceiver UP or Down as instructed by the data link layer.
O_rate	Output	2 bits	Controls the link data rate.
O_reset	Output	1 bit	Reset the link
O_Block_align	Output	1 bit	Ask the PYH part to perform the block alignment, used in GEN 3.0 and above.
I_Rcv_Detected	Input	1 bit	Receiver Detected.
I_RX_Valid	Input	1 bit	Symbol lock achieved, and the data received is now valid.
I_RX_EIdle	Input	1 bit	Asserted whenever the receiver is detected to be in electrical idle.
I_PhyStatus	Input	1 bits	Asserted when certain PHY functions has been completed, such as power management, receiver detection, and rate change.

4. PIPE / SERDES Interface

• Definitions

- **PIPE**: Stands for ***P***HY ***I***nterface ***P***CI ***E***xpress Architecture, which enables the interaction between the MAC layer (logical part of physical layer) and PHY (electrical part of physical layer)

- **SerDes**: Stands for ***Ser***ializer/ ***De-s***erializer, and it also defines the interface between the logical and electrical part of the physical layer, but with more functions included in the logical part than those included in the case of PIPE.

• Design Specific Implementation

- In the design presented, the support for both interfaces is included, with an assumption that the SERDES support is only for Higher GENs, but the PIPE support is for all GENs.
- Consequently, the additional functions Included in the MAC are only implemented for the case of operating at high Data Rate. (i.e: 128/130 bits encoding/ decoding is implemented ,while the 8/10 bits encoding/decoding is assumed to be performed by the PHY).

• Function Discrepancies between PIPE and SERDES

Function	PIPE	SERDES
128/130 bits encoding/ decoding	Not Included in MAC	Included in MAC
Block Alignment	Not Included in MAC	Included in MAC
Control of Elastic Buffer including SKP removal	Not Included in MAC	Included in MAC

- **Signals Discrepancies between PIPE and SERDES**

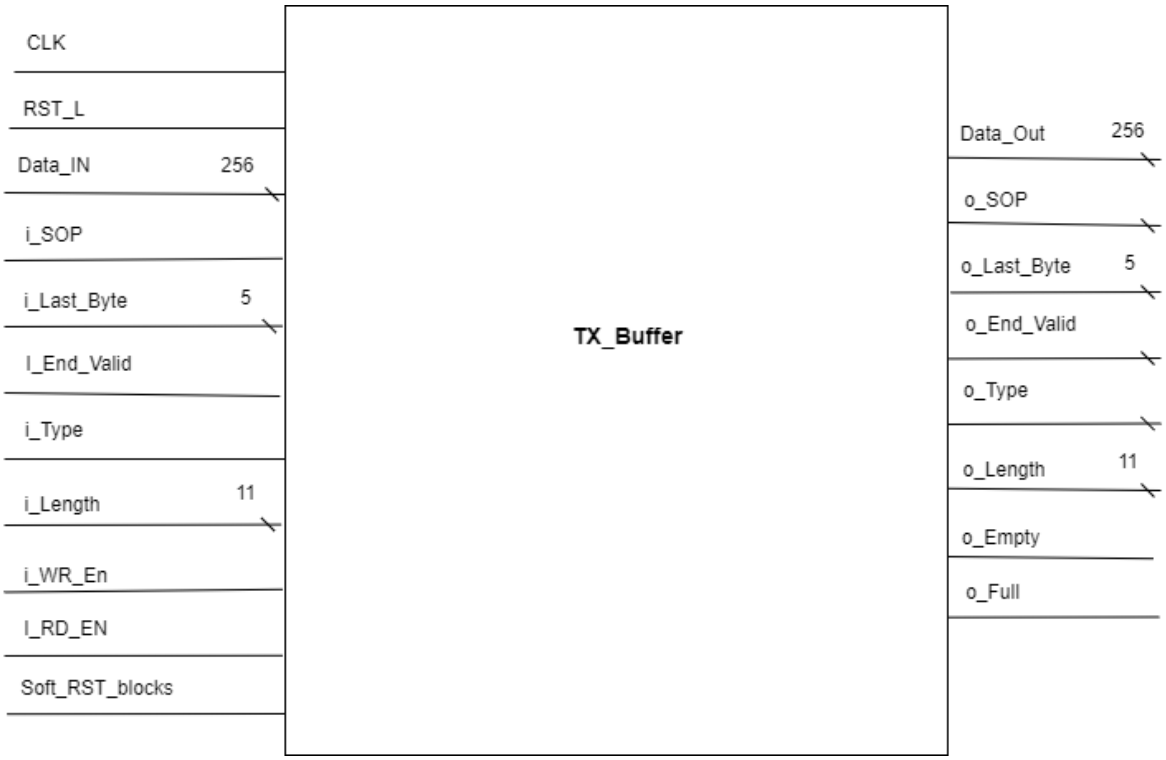
Signal	Width	Description	PIPE	SERDES
CLK	1 bit	Local system CLK	✓	✓
Rx CLK	1 bit	Bit wise recovered CLK in RX		✓
TX_Data	256 bits	Output from TX 1 scrambled symbol output from each lane in TX	✓	
Sync_Data	256 bits	Output from TX 1 scrambled and sync_header added symbol from each lane		✓
TX_Start_Block	32 bits	Output from TX Asserted with symbol zero in Data Block	✓	
TX_Sync_Header	2*32 bits	Output from TX Holds the 2 bits sync header that should be added at the start of block in each lane	✓	
RX_Start_Block	32 bits	Input to RX Asserted with symbol zero in Data Block	✓	
RX_Sync_Header	2*32 bits	Input to RX Holds the 2 bits sync header that identifies the type of current input block	✓	
RXValid	32 bits	Input to RX Indicates the Validity of Symbol on each lane in low GENs (symbol and bit lock achievement)	✓	
RX_Data_ Valid	32 bits	Input to RX Indicates the validity of Symbol on each lane in High GENs	✓	

O_St_Detect	32 bits	Output to PHY Asserted when there is a need to start the detection of receiver on the lanes	✓	✓
O_E_Idle	32 bits	Output to PHY Forces TX output to electrical Idle	✓	✓
O_Pwr_States	3 * 32 bits	Output to PHY Power the transceiver UP or Down as needed in the different states.	✓	✓
O_rate	2 bits	Output to PHY Controls the link data rate.	✓	✓
I_Rcv_Detected	32 bits	Input to LTSSM Indicates the detection of a receiver on the lane	✓	✓
I_RX_EIdle	32 bits	Input to LTSSM Asserted whenever the receiver is detected to be in electrical idle.	✓	✓
I_PhyStatus	1 bit	Input to LTSSM Asserted when certain PHY functions has been completed, such as power management, receiver detection, and rate change.	✓	✓

❖Detailed Description

1. Transmitter Blocks

1.1. Tx Buffer



- **Description**

  - This Buffer is a Sync FIFO that required to allow for continuous flow of data from DL,L even when physical layer is busy sending OSs, and to accommodate for the halting resulting from 128b/130b encoding scheme.
  - The size of Buffer contain width (32 Byte (Data) + 19 bits (packet indicators)) and depth 16 location that calculated according to 16 cycle for sending SKP Symbols as no need to stop DLL this long period although we can halt them in other situations if the Buffer is full.

- **Working Mechanism**

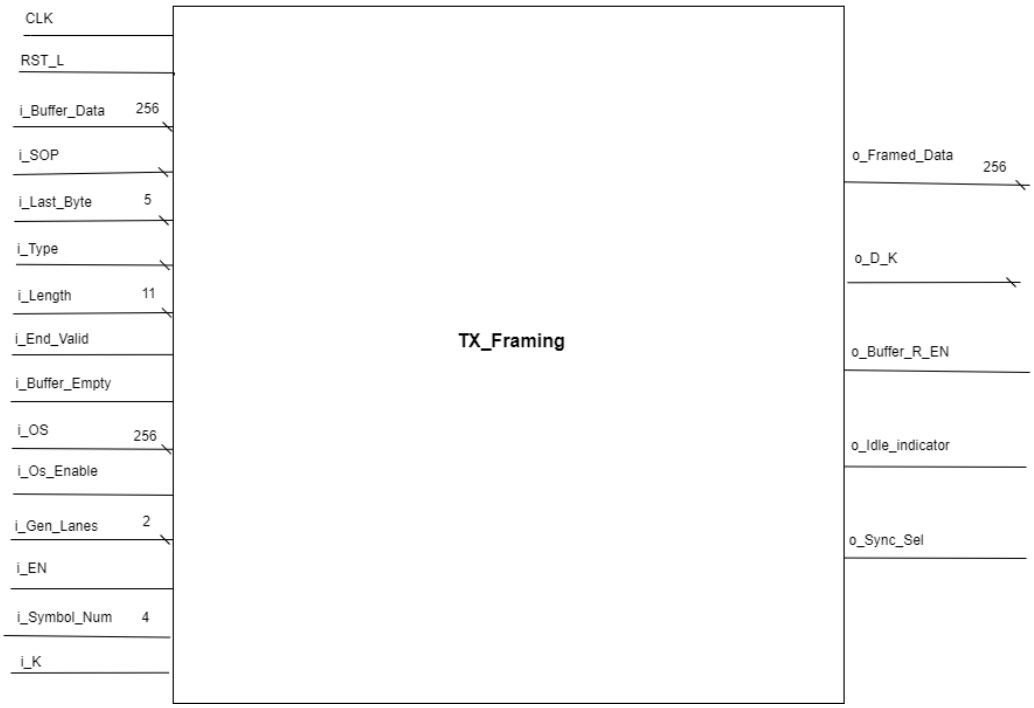
  - Each location in the buffer could contain 32 byte valid data along with some control signals as provided by DLL, which are **i_SOP, i_Last_Byte, i_End_Valid, i_Type and i_Length .**
  - There exists a read pointer that is controlled by the Tx internal logic and a write pointer that is controlled by the data link layer
  - To enable reading from the buffer a read enable signal (**i_RD_EN**) should go high and (**i_WR_EN**) to enable writing in the Buffer.
  - Buffer outputs empty (**o_Empty**) signal to indicate that no more TLPs or DLLPs are ready.
  - Buffer outputs full signal (**o_Full**) to stop DLL from inserting more data into the buffer to avoid overflow and loss of data.

- **Ports**

Name	Direction	Size	Description
CLK	Input	1 bit	TX CLK
RST	Input	1 bit	Global Reset signal
i_Data	Input	256 bits	Input Data bus from Data-Link layer.
i_SOP	Input	1 bit	Input from DLL 1- Start indicator from Data-Link layer. Goes high when I_Date represents a start of a new packet.
i_Last_Byte	Input	5 bits	Input from DLL 1- End of packet pointer. Indicates which byte in the 32 bytes of the current I_Data represents the end of the packet.
i_End_Valid	Input	1 bit	Input from DLL - Goes high when packet ends by the current I_Data to validate reading I_EOP
i_Type	Input	1 bit	Input from DLL 1- Indicates whether I_ Data is a TLP or DLLP. 2- A value of 1 indicates a TLP A value of 0 indicates a DLLP.
i_Length	input	11 bits	Input from DLL -Represents the total size of the TLP that I_Data is part of, to be considered in framing tokens.

i_WR_EN	Input	1 bit	Input from DLL -Enables inserting new data into the buffer.
i_RD_EN	input	1 bit	Input from TX_Framing to enable reading the next location.
Soft_RST_blocks	input	1 bit	Input from LTSSM to reset the Buffer in case Error Handling.
o_Data	output	256 bits	Output Data location of the buffer.
o_SOP	output	1 bits	Output start indicator for each location.
o_Last_Byte	output	5 bits	Output end placement indicator for each output location.
o_End_Valid	output	1 bits	Validates o_Last_Byte for each output location
o_Type	output	1 bits	1- Output packet type for each location 2- Indicates whether Data_Out is a TLP or DLLP. 3- A value of 1 indicates a TLP A value of 0 indicates a DLLP.
o_Length	output	11 bits	Output packet length for each location.
o_Empty	output	1 bit	Goes high when buffer is empty
o_Full	output	1bit	Goes high when buffer is full

1.2. Tx_Framing



- **Description**

  - This block is the main data supplier to all blocks in the TX architecture. It is responsible for framing the packets received from DLL according to certain framing rules. It also decides when to send the scheduled OSs so as not to interrupt a data block or a packet that has been started already. It sends logical idles when the buffer that contains received packets from DLL is empty and there are no scheduled OSs.

- **Working Mechanism**

  - It inserts STP,SDP tokens to the input data (**Buffer_Data**) according to the value of (**SOP, i_Last_Byte ,End_Valid and Length, Type**) signals
  - Since, the block itself adds additional bytes due to framing tokens, buffering (2 bytes) is needed inside in case TLP packets so, we add Framing Buffer block.
  - When the buffer is empty, as indicated from (**i_Buffer_Empty**) signal, there are no ordered sets to be sent, as indicated from the

(**i_Os_Enable**) signal, and no remaining bytes are buffered inside, it starts sending logical idles.

- When operating at 8GT/S or higher, as indicated from the (**i_Gen_Lanes**) signal**,** and when the (**i_Os_Enable**) signal is asserted, it uses the (**i_Symbol_Num**) signal to decide whether it can send the EDS token to end the current data stream and start sending the scheduled OS, or it should keep on sending data or idles until the **(i_Symbol_Num = 15)** in case 32 lane and **(i_Symbol_Num = 12)** in case 1 lane.
- It asserts the (**o_Sync_Sel**) signal when sending OSs to be used by the Sync_Logic block and OS_Creator block ,Since the assertion of (**i_Os_Enable**) does not necessarily mean that OSs transmission is to be started right away.
- The input (**i_EN**) signal is needed to stop the flow of output data there is a back pressure due to the accumulation of bits in the sync_logic block and then it should stop reading any more data from TX_Buffer by de-asserting the (**i_Buffer_R_EN**) signal.

- **Ports**

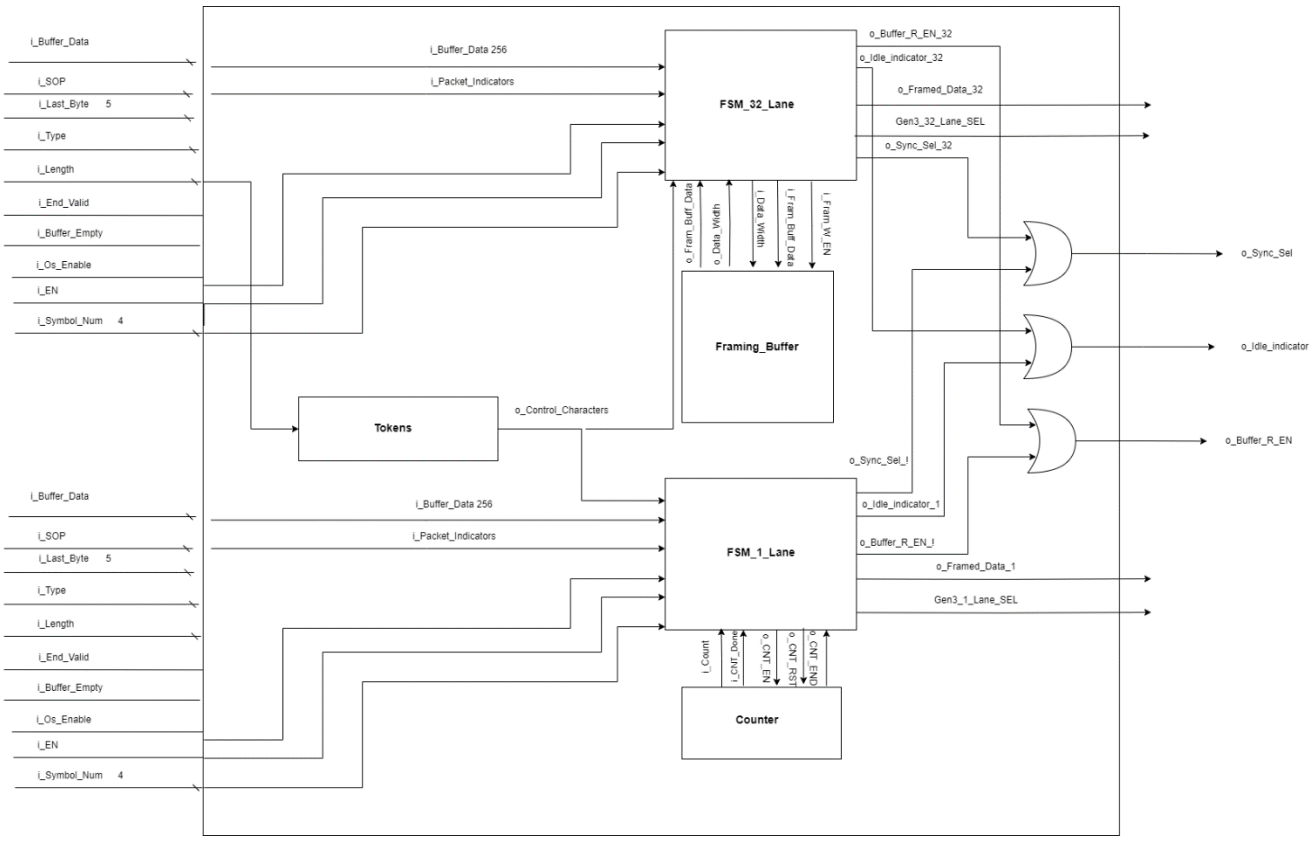
Name	Direction	Size	Description
CLK	Input	1 bit	TX CLK
RST	Input	1 bit	Global Reset signal
i_Buffer_Data	Input	256 bits	Input Data to from Tx_Buffer
i_SOP	Input	1 bit	-Start indicator for each 32 bytes of Buffer_Data
i_Last_Byte	Input	5 bits	End indicator for last Byte in last location of packet of Buffer Data.
i_End_Valid	Input	1 bit	Validates o_Last_Byte for each 32 bytes of Buffer_Data
i_Type	Input	1 bit	packet type for each 32 bytes of Buffer_Data
i_Length	input	11 bits	packet length for each 32 bytes of Buffer_Data
i_K	Input	1 bit	Indicator From LTSSM for Control characters of OS.
i_Buffer_Empty	Input	1 bit	Goes high when buffer is empty
i_OS	Input	256 bits	OS symbols as provided from LTSSM
i_Os_Enable	Input	1 bits	Goes high when an OS is scheduled
i_Gen_Lanes	input	2 bits	Indicates the configured generation and the number

			of lanes after finishing link training 1- 00 (Gen1 or Gen2 and 32 lanes) 2- 10 (Gen3 or higher and 32 lanes) 3- 01 (Gen1 or Gen2 and 1 lane) 11 (Gen3 or higher and 1 lane)
i_EN	Input	1bit	An enable signal to the whole logic
i_Symbol_Num	Input	4bits	Counter for each Symbol in the block from Sync Logic.
o_Framed_Data	Output	256bits	Output data after framing to be sent to byte_stripping
o_Buffer_R_EN	Output	1bit	Enables reading more data from TX_Buffer
o_D_K	Output	1bit	Indicate that the output is Data or Control.
o_Idle_indicator	Output	1bit	Goes high when starting sending logical idle
o_Sync_Sel	Output	1bit	Goes high when transitioning to send OSs instead of Data so that it could be used by Sync_Logic

- **Implementation**

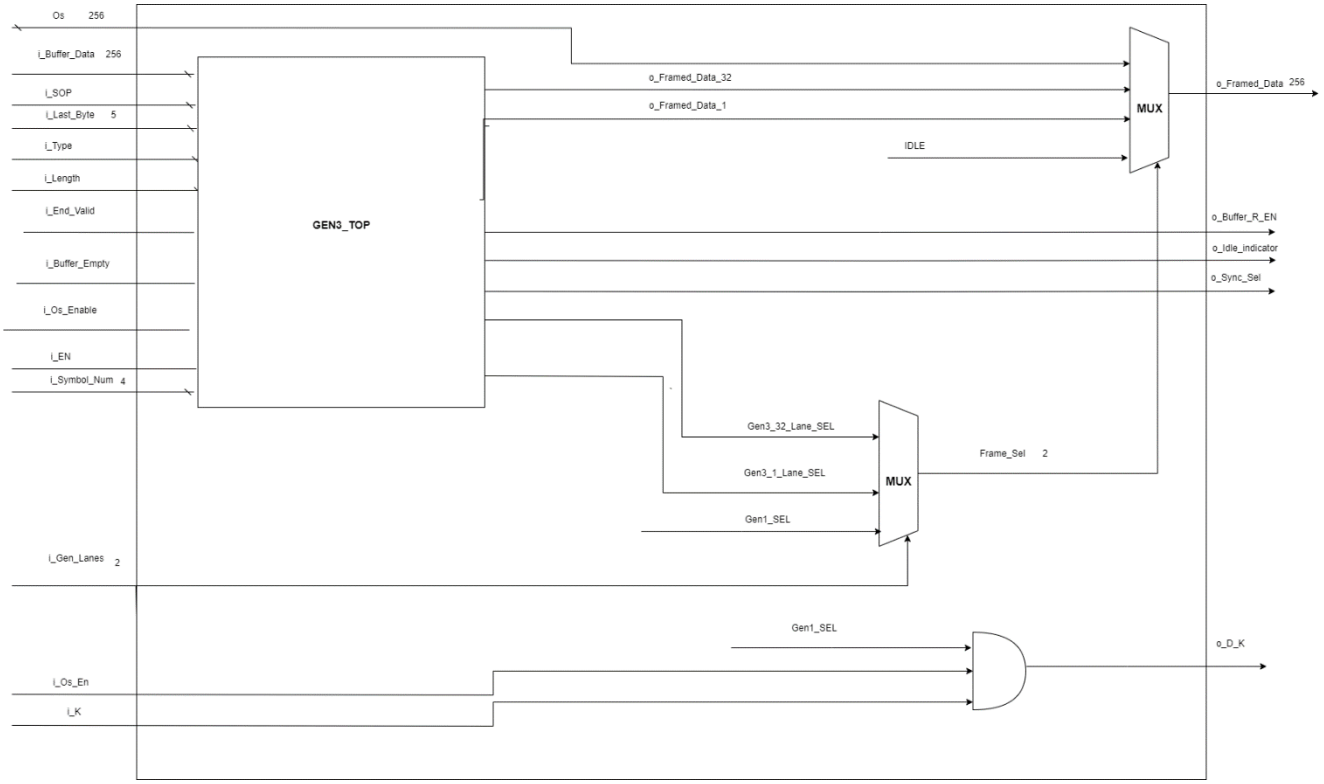
  - **TX_Framing TOP**

    - TX Framing in our design Supports Gen1,2 in Training sequence only and Gen 3,4,5 in training sequence and data transmission in case 32 lane and 1 lane so we needed multiplexer to out (**Fram Sel**) from Gen3_32 lane or Gen_3_1 lane or Gen1,2 according to (**i_Gen_Lanes**).
    - We need another multiplexer to out (**o_Framed_Data**) from Framed_Data_32 or Framed_Data_1 or IDLES or OS according to (**Fram_Sel**).
    - (**o_D_K**) signal is asserted in case Gen1,2 and there are OS to be sent that is control characters.
    - And these inernal outputs controlled and out by GEN3_TOP

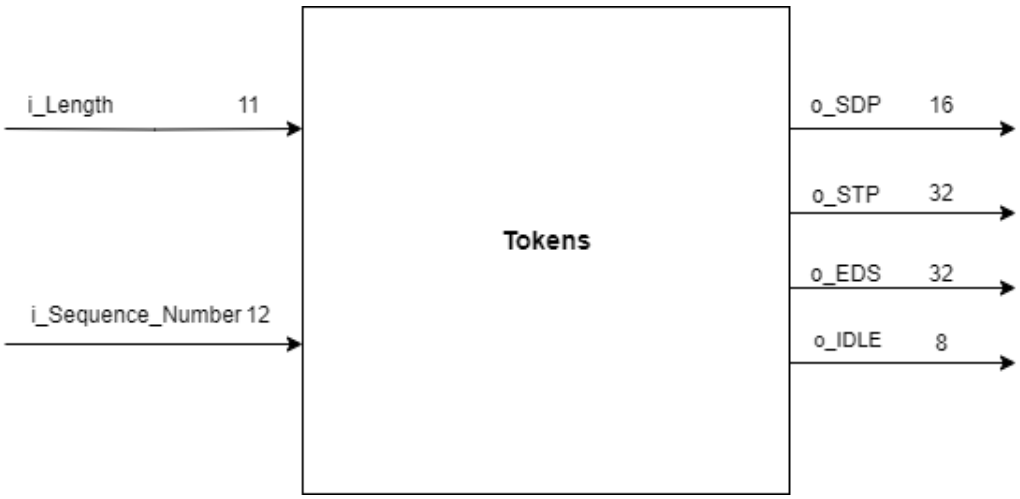


○ Gen3 TOP

- Gen3_Top form the Tokens and control the output framed data to frame the packets according to packet indicators and logic of each block and is divided into 2 parts as first part to control and out the framed data in 32 lane and second part to control and out the data in 1 lane but out the read enable, IDLE Indicator and sync sel by Oring each of them in 1 lane and 32 lane



○ Tokens



- **Description**

- Tokens is required to form the tokens of packets (STP – SDP – EDS – IDLES) as these tokens is divided into fields that need some logic to calculate it like Frame CRC or Frame Parity and so on.

- **Working Mechanism**

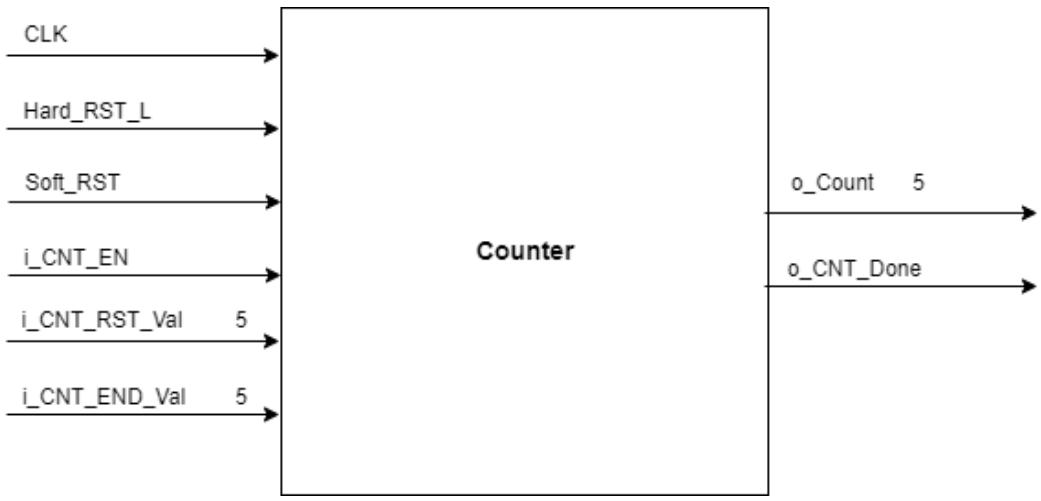
- Tokens take the inputs (**i_Length**) and (**i_Sequence_Number**) as a part of Received data from TX_Buffer (first 12 bits) and send ready Tokens to be used in FSMs.
- Tokens Form each token by concatenating the header with other tokens field that need some of logic to calculate it like Frame CRC that is XORING the length or concatenating the header only like SDP Token.

- **Ports**

Name	Direction	Size	Description
i_Length	input	11 bits	packet length for each 32 bytes of Buffer_Data
i_Sequence_Numbr	Input	12 bits	Part of the input received data from TX_Buffer and is used to form STP Token

O_STP	Output	32 bits	STP Token to frame TLP packets and input to FSMs of Frame.
O_SDP	Output	16 bits	SDP Token to frame DLLP packets and input to FSMs of Frame.
O_EDS	Output	32 bits	EDS Token to End the block and send OS in the next block and input to FSMs of Frame.
O_IDLE	Output	8 bits	IDLE Token to send when there are no Data or OS to be sent.

- ## Counter



- ## Description

  - Counter is required to communicate with FSM 1 lane as we need to count each symbol of Tokens to start to send Data or OS or count each symbols of length of packets to finish packet and start new packet.

- ## Working Mechanism

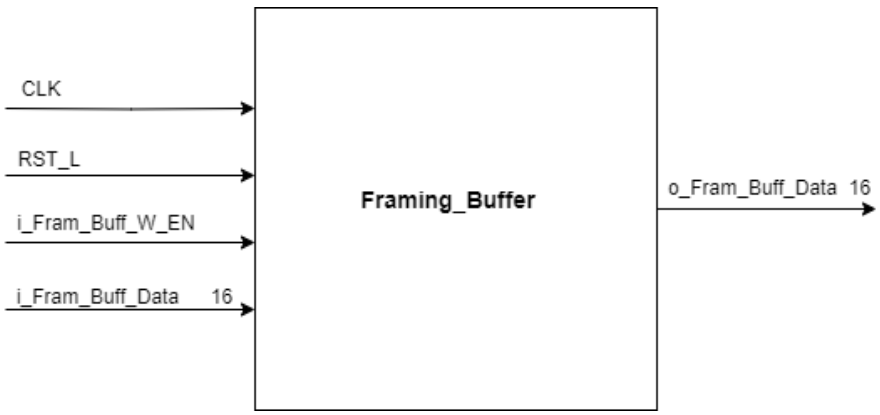
  - Counter is used in States STP and SDP and EDS as it take its inputs according to each state such that (**i_CNT_END_Val**) to be equal with last symbol of token and out (**o_CNT_Done**) when reach to this count or It take the count enable if it doesn't reach to last count.

- Counter can start counting from certain count when using (**i_CNT_RST_Val**) to start with it and use the soft reset.

- **Ports**

CLK	Input	1 bit	TX CLK
Hard_RST	Input	1 bit	Global Reset signal
Soft_RST	Input	1 bit	Enable to start count from certain value.
i_CNT_EN	Input	1 bit	Enable to continue counting.
i_CNT_END_Val	Input	5 bits	Value of the Last count to out the count Done.
i_CNT_RST_Val	input	5 bits	Start value of counter in case Soft RST.
o_CNT_Done	Output	1 bit	End indicator of the Last count.
o_Count	Output	5 bits	Indicate the current count.

○ Framing Buffer



- **Description**

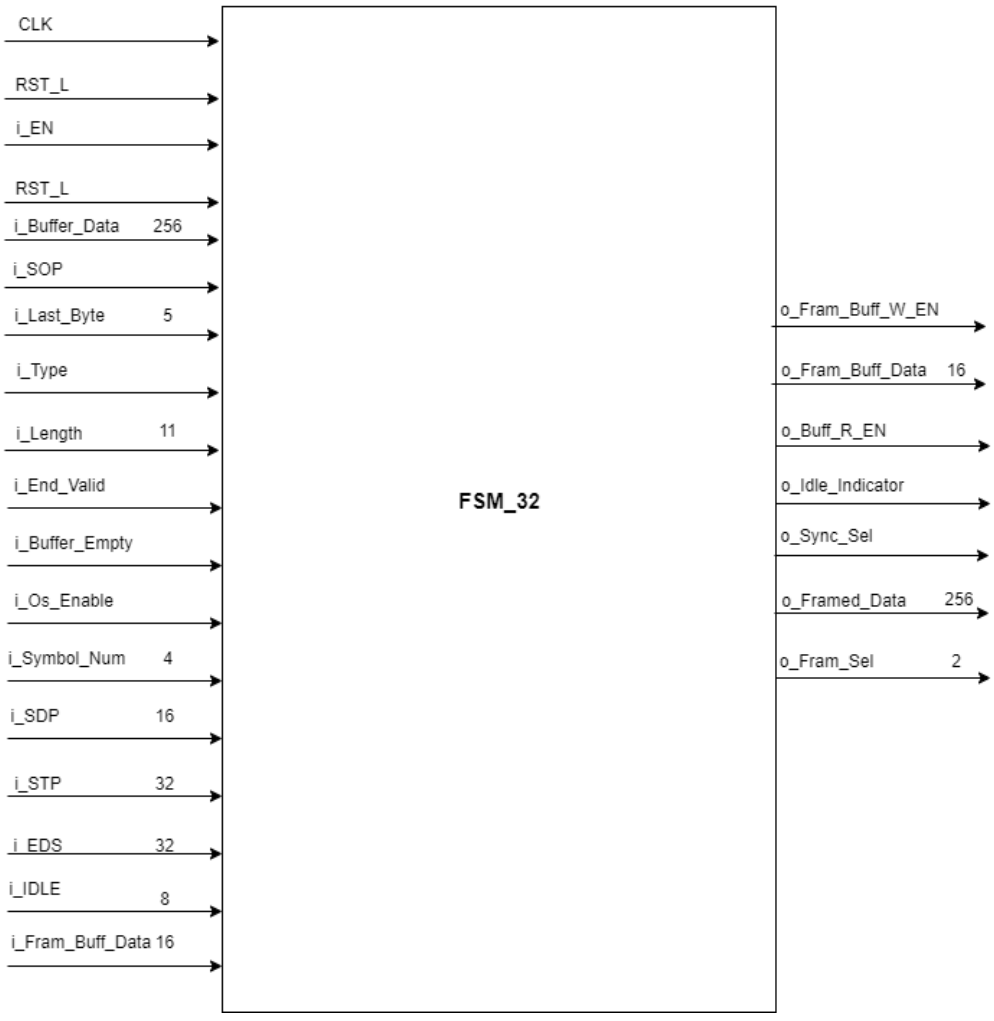
- Framing Buffer is required to store the Last 2 bytes of Data due to the 2 bytes of Tokens that we add at the start of TLP packet so we need the Buffer.

- **Working mechanism**

  - We enable the writing operation using sig (i_Fram_Buff_W_EN) in case Start of TLP Packet with Length > 6 DW that sent on more than location.
  - After storing the 2 bytes the next cycle we concatenate the data stored in buffer with new location and store another 2 bytes if exist.
  - It's required only in 32 lane states as in one lane we send byte by byte.

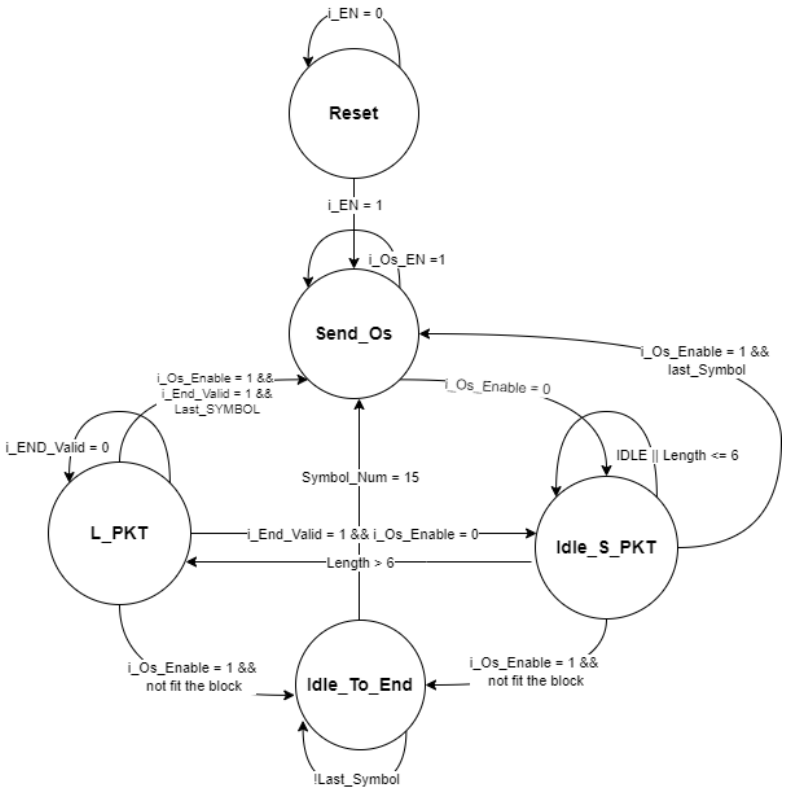
- **Ports**

CLK	Input	1 bit	TX CLK
RST_L	Input	1 bit	Global Reset signal
i_Fram_Buff_W_EN	Input	1 bit	Enable writing data
i_Fram_Buff_Data	input	16 bits	Writing Data that stored in Buffer.
o_Fram_Buff_Data	output	16 bits	Reading Stored Data from Buffer.



- **Description**

  - FSM_32 is required to control Framing Data in case 32 lane as it's responsible to out the final Form of data framed by Tokens and out the Fram Sel to out Data or out OS's in case there's OS's need to be sent or IDLES in case there's no ready Data nor OS's.



- **Working mechanism**

  - FSM_32 takes the ready Tokens from Tokens module and concatenate the Tokens with Data from TX Buffer and store the last 2 bytes in Framing Buffer in case of Large packet and read them and concatenate in next cycle and output (**o_Buff_R_EN**) after reading each location and out (**o_IDLE_Indicator**) , (**o_Sync_Sel**) to Sync logic and timer according to the tates as shown below

  - **Reset**: The initial State, and remain in this state until LTSSM enable the TX_Framnig block so, we move to Send_OS State and all outputs take the default values.
  - **Send_Os**: in this state, we send OSs that received From LTSSM until i_Os_Enable = 0 so no ready OS to be sent so, we move to Idle_S_PKT to send Data or IDLES.

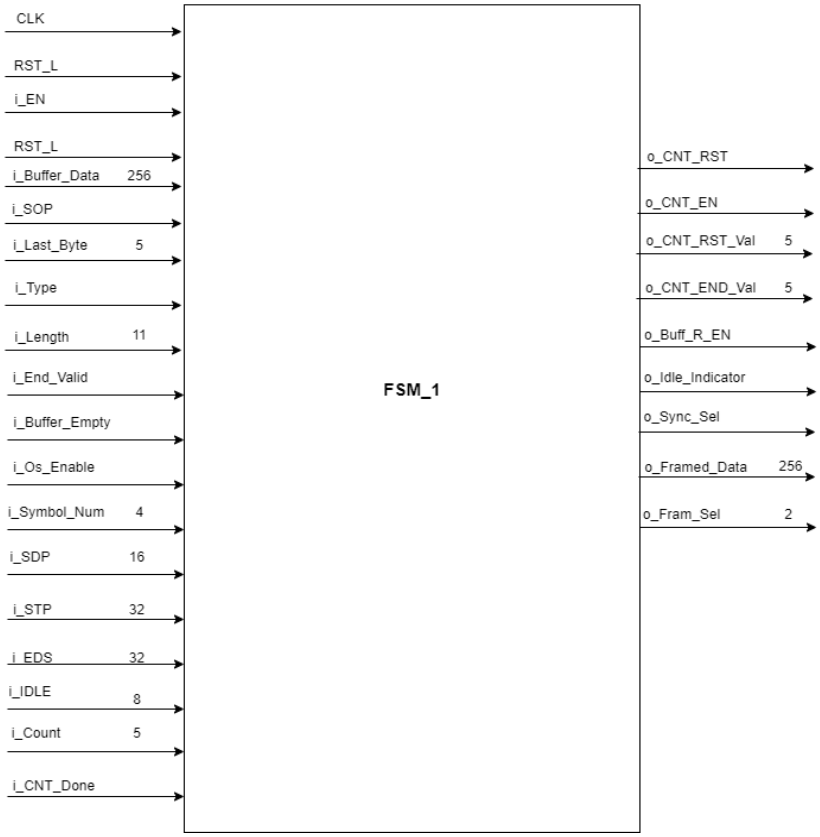
- **Idle_S_PKT**: in this state, we send IDLES if no ready data in TX Buffer or Send small packet that can be sent in one loaction and if the packet is sent on more than one location we send the first location then move to (**L_PKT)** state and we move to (**Send_Os)** state when (i_**Os_Enable** = 1) and this is the last symbol or move to (**Idle_To_End**) state if the next packet not fit the block.
- **L_PKT**: in this state, we send Large packets that can be sent in more than location and move to (**Send_Os)** if there is ready ordered sets and this is the last symbol and fit in the block or move to (**Idle_To_End)** state if the the next packet not fit the block or move to (**Idle_S_PKT)** if the packet has been finished and there's no ready OS's.
- **Idle_To_End**: in this state, we send Idle until Last symbol we Frame it with EDS token and move to (**Send_Os)** state.

- **Ports**

CLK	Input	1 bit	TX CLK
RST	Input	1 bit	Global Reset signal
i_Buffer_Data	input	256 bits	Input Data to from Tx_Buffer
i_SOP	input	1 bit	-Start indicator for each 32 bytes of Buffer_Data
i_Last_Byte	input	5 bits	End indicator for last Byte in last location of packet of Buffer Data.
i_End_Valid	input	1 bit	Validates o_Last_Byte for each 32 bytes of Buffer_Data
i_Type	input	1 bit	packet type for each 32 bytes of Buffer_Data
i_Length	input	11 bits	packet length for each 32 bytes of Buffer_Data
i_Buffer_Empty	input	1 bit	Goes high when buffer is empty
i_Fram_Buff_Data	input	16 bits	Data Stored in Framing Buffer.
i_STP	input	32 bits	STP Token to frame TLP packets.
i_SDP	input	16 bits	SDP Token to frame DLLP packets.
i_EDS	input	32 bits	EDS Token to End the block and send OS in the next block.

i_IDLE	input	8 bits	IDLE Token to send when there are no Data or OS to be sent.
i_Os_Enable	input	1 bits	Goes high when an OS is scheduled
i_EN	input	1bit	An enable signal to the whole logic
i_Symbol_Num	input	4bits	Counter for each Symbol in the block from Sync Logic.
o_Buffer_R_EN	output	1bit	Enables reading more data from TX_Buffer
o_Idle_indicator	output	1bit	Goes high when starting sending logical idle
o_Sync_Sel	output	1bit	Goes high when transitioning to send OSs instead of Data so that it could be used by Sync_Logic
o_Framed_Data	output	256 bits	Output data after framing to be sent to byte_stripping
o_Fram_Sel	output	2 bits	Output sel to select the output Frame Data and encoded as: 00 : IDLE 01: OS 10: FRAM_32 11: FRAM_!
o_Fram_Buff_W_EN	output	1 bit	Enable writing data in Framing Buffer
o_Fram_Buff_Data	output	16 bits	Writing Data that stored in Buffer.

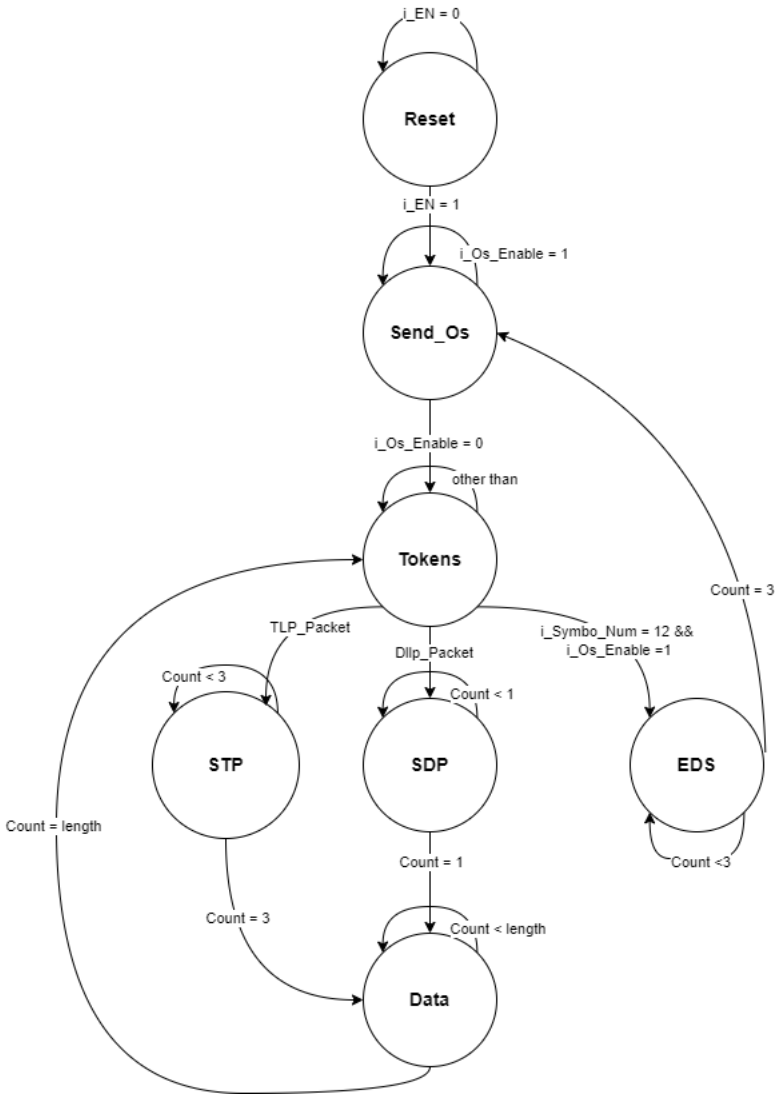
○ FSM_1



- **Description**

  - FSM_1 is required to control Framing Data in case 1 lane as it's responsible to out the final Form of data framed by Tokens and out the Fram Sel to out Data or out OS's in case there's OS's need to be sent or IDLES in case there's no ready Data nor OS's.

- **Working mechanism**



- FSM_1 takes the ready Tokens from Tokens module and concatenate the Tokens with Data from TX Buffer and communicate with the counter to count each Token that take 4 cycle to be sent for example and output (**o_Buff_R_EN**) after reading each location and out (**o_IDLE_Indicator**) , (**o_Sync_Sel**) to Sync logic and timer according to the states as shown below:

- **Reset**: The initial State, and remain in this state until LTSSM enable the TX_Framnig block so, we move to Send_OS State and all outputs take the default values.

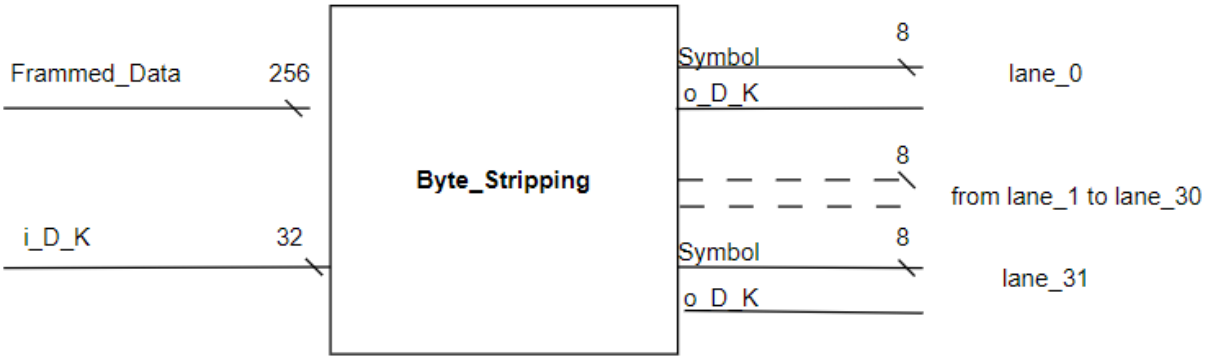
- **Send_Os**: in this state, we send OSs that received From LTSSM until i_Os_Enable = 0 so no ready OS to be sent so, we move to (**Tokens**) to send Data or IDLES.
- **Tokens**: in this state, we calculate the logics of packet indicators and move to the next state according to Type of packet as we move to (**STP**) if TLP packet or move to (**SDP**) if DLLP packet or move to (**EDS**) if there are Os's to be sent and symbol num = 12.
- **STP**: in this state, we send STP token in 4 cycles so we need counter to count the cycles and if count = 3 we move to send (**Data**) of TLP packet.
- **SDP**: in this state, we send SDP token in 2 cycles so we need counter to count the cycles and if count = 2 we move to send (**Data**) of DLLP packet.
- **EDS**: in this state, we send EDS token in 4 cycles so we need counter to count the cycles and if count = 3 we move to (**Send_Os**).
- **Data**: in this state, we send the data of any packets until the packet finished and move to (**Tokens**) to check them again.

- **Ports**

CLK	Input	1 bit	TX CLK
RST	Input	1 bit	Global Reset signal
i_Buffer_Data	input	256 bits	Input Data to from Tx_Buffer
i_SOP	input	1 bit	-Start indicator for each 32 bytes of_Buffer_Data
i_Last_Byte	input	5 bits	End indicator for last Byte in last location of packet of Buffer Data.
i_End_Valid	input	1 bit	Validates o_Last_Byte for each 32 bytes of Buffer_Data
i_Type	input	1 bit	packet type for each 32 bytes of_Buffer_Data
i_Length	input	11 bits	packet length for each 32 bytes of_Buffer_Data
i_Buffer_Empty	input	1 bit	Goes high when buffer is empty
i_CNT_Done	input	1 bit	End indicator of the Last count.
i_Count	input	5 bits	Indicate the current count.
i_SDP	input	16 bits	SDP Token to frame DLLP packets.
i_EDS	input	32 bits	EDS Token to End the block and send OS in the next block.

i_IDLE	input	8 bits	IDLE Token to send when there are no Data or OS to be sent.
i_Os_Enable	input	1 bits	Goes high when an OS is scheduled
i_EN	input	1bit	An enable signal to the whole logic
i_Symbol_Num	input	4bits	Counter for each Symbol in the block from Sync Logic.
o_Buffer_R_EN	output	1bit	Enables reading more data from TX_Buffer
o_Idle_indicator	output	1bit	Goes high when starting sending logical idle
o_Sync_Sel	output	1bit	Goes high when transitioning to send OSs instead of Data so that it could be used by Sync_Logic
o_Framed_Data	output	256 bits	Output data after framing to be sent to byte_stripping
o_Fram_Sel	output	2 bits	Output sel to select the output Frame Data and encoded as: 00 : IDLE 01: OS 10: FRAM_32 11: FRAM_!
o_CNT_RST	output	1 bit	Enable to start count from certain value.
o_CNT_EN	output	1 bit	Enable to continue counting.
o_CNT_END_Val	output	5 bits	Value of the Last count.
o_CNT_RST_Val	output	5 bits	Start value of counter in case Soft RST.

1.3. Byte Stripping



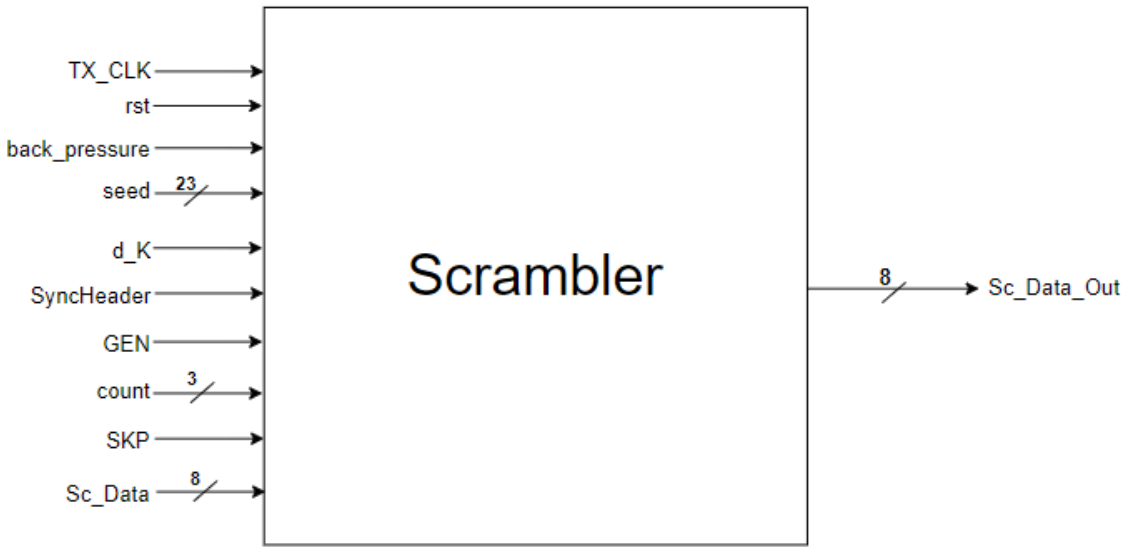
- **Description**

  - Byte striping block is needed for wide links (the port supports more than one Lane). Striping is the process of routing each outgoing character in a character stream onto a consecutive Lane. During the Link training process, the number of Lanes that are used is configured according to what both of the devices sharing the Link support.
  - The supported lane widths in our implantation are 1 lane and 32 lanes. Thus, N is either 1 or 32.

- **Ports**

Name	Direction	Size	Description
Frammed_Data	**Input**	**256 bits**	**output from Tx_Framing**
I_D_K	**Input**	**32 bits**	**output from Tx_Framing (Data_Control for Gen1,2)**
Symbol	**Output**	**8 bits**	**input to scrambler on each lane.**
O_D_K	**Output**	**1 bit**	**input to scramble (Gen1,2) on each lane.**

1.4. Scrambler



- **Description**

  - The scrambler plays a significant role in avoiding the creation of pure tones on the link, which results from repetitive patterns. The implementation of the Scrambler is LFSR based. In case of GEN 3 and higher (128b/130b encoding), the scrambler needs to address some issues that were handled by the encoder of lower generations (8b/10b) such as: DC balance, and enough transition edges at the receiver to recover the transmitter clock.

- **Working mechanism**

  - The scrambling rules implemented for lower generations are:

    a. Scrambling is never applied to the 'K' characters, and the characters within the ordered sets such as TS1s, TS2s, EIOSs,… And this is to ensure that they will be recognized by the receiver even if the scramblers get out of sequence.
    b. The COM character is used to reinitialize the LFSR to FFFFh at both transmitter and receiver.
       - The scrambling rules implemented for higher generations are:

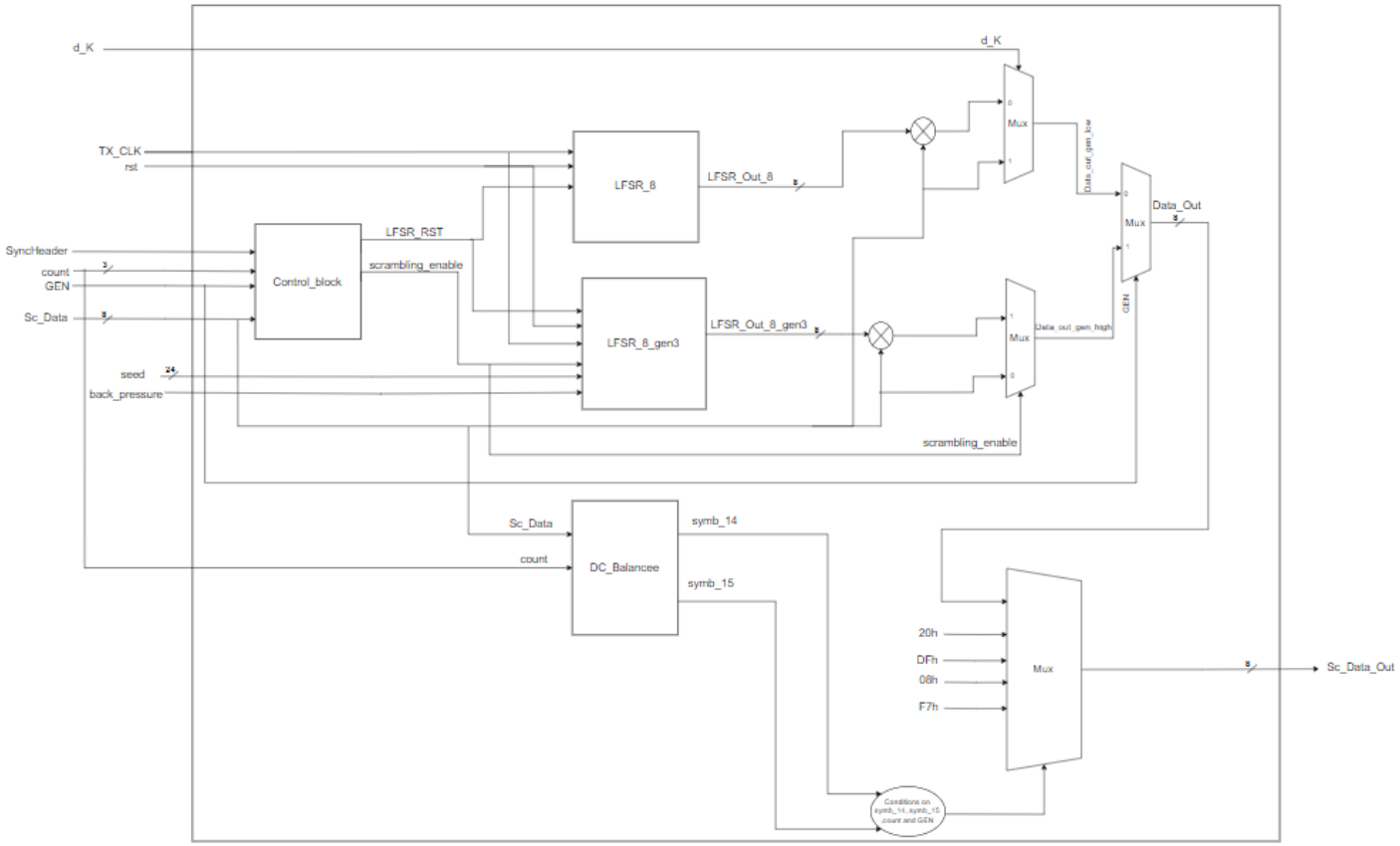
a. The transmitter and receiver LFSR are reset with the last symbol of the EIEOS.
b. TS1s and TS2s ordered sets have a special treatment: symbol 0 bypasses scrambling, symbols 1 to 13 are scrambled and symbols 14 and 15 may be scrambled or not depending on the state of the DC balance.
c. All symbols of the ordered sets: FTS, SDS, EIOS, EIEOS, and SOS bypass scrambling, but they advance the LFSR except for the SOS.
d. All data block symbols are scrambled and advance the LFSR.
e. Symbols are scrambled in little- endian order.
f. The seed value for a per lane LFSR depends on the lane number assigned during the link training.
g. The scrambler cannot be disabled as in lower GENs, as the link would not operate reliably without, as it handles the DC balance and the density of transitions for clock recovery.

- **Ports**

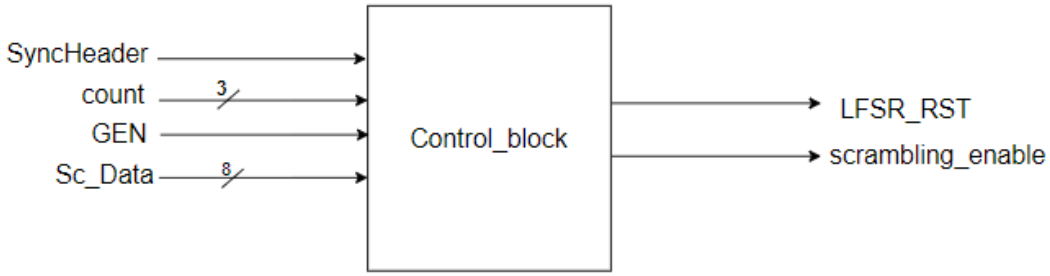
Name	Direction	Size	Description
TX_CLK	Input	1 bit	Transmitter clock
Rst	Input	1 bit	System asynchronous reset
d_K	Input	1 bit	Indicates whether the symbol is data character or control character. 0: data character 1: (K) control character
SyncHeader	Input	1 bit	Indicate the block type for GEN3: 1'b0: Ordered Set 1'b1: Data
count	Input	3 bits	Indicate the current symbol number in the block.
Sc_Data	Input	8 bits	The input data in each lane after the TX framing.
GEN	Input	1 bit	The current generation.
seed	Input	24 bits	The initial values of the LFSR coming from the LTSSM.
back_pressure	Input	1 bit	The back pressure coming from the buffer logic.
Sc_Data_Out	Output	8 bits	The scrambler data which is the output of the LFSR xored with the input data or the original data if it bypassed the LFSR.

- **Implementation**

  - **Scrambler Top**



  - **Scrambler Control block**



- **Description**

  - The control logic is responsible for the resetting of the LFSR as well as the enabling signal for the LFSR at higher generations (LFSR_8_gen3). It doesn't control the scrambling enable of the lower generation as in the lower generations the d_K signal (data or control) specifies whether the input data is scrambled or not unlike the case in higher generations which requires a set of restricting rules.
  -

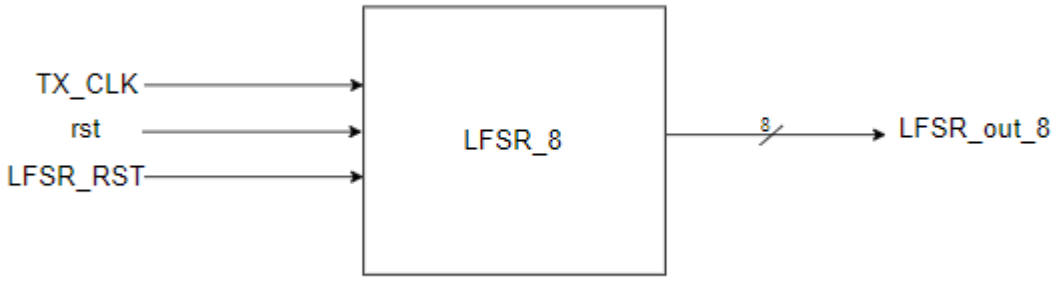
- **Working Mechanism**

  - It controls whether the LFSR of generation 3 and above is allowed to scramble the data according to the specified rules or to bypass the data as it is.
  - Moreover, it controls whether the LFSR is reset or not. For instance, if a COM character appears at any lane the LFSR_8 (for lower generations) , LFSR_RST will reset the registers inside the LFSR_8. While at higher generations if the last symbol of EIOS appeared the LFSR_RST will reset the registers inside the LFSR_8_gen3.

- **Ports**

Name	Direction	Size	Description
SyncHeader	Input	1 bit	Indicate the block type for GEN3: 1'b0: Ordered Set 1'b1: Data
Count	Input	3 bits	Indicate the current symbol number in the block.
Sc_Data	Input	8 bits	The input data in each lane after the TX framing.
GEN	Input	1 bit	The current generation.
LFSR_RST	Output	1 bit	For resetting the LFSR when a COM character appears or at the end of EIOS.
scrambling_enable	Output	1 bit	For enabling the scrambling in higher generations.

- **LFSR_8 Block**



- **Description**

  - The name LFSR_8 is created as each lane has an incoming data of 8 bits and these bits are xored with the output of the LFSR which is the advanced 8 bits of the LFSR.
  - LFSR_8 block is used only for lower generations (Gen 1 and 2).

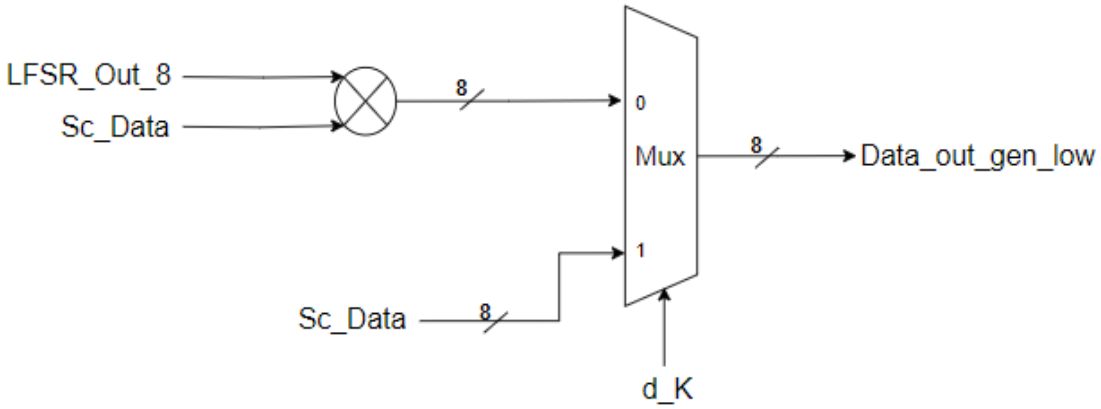
- **Working Mechanism**

  - It's made of a 16-bit Linear Feedback Shift Register (LFSR) with feedback points that implement the following polynomial: $\boldsymbol{G(X) = X^{16} + X^5 + X^4 + X^3 + 1}$
  - Since the input data is 8 bits, it needs 8 output bits to be xored with, however the LFSR outputs 1 bit each clock. The issue can be solved by advancing the LFSR 8 times each cycle, thus the input data is xored each cycle with the 8 outputs from the LFSR.

- **Ports**

Name	Direction	Size	Description
LFSR_RST	Input	1 bit	For resetting the LFSR when a COM character appears.
TX_CLK	Input	1 bit	Transmitter clock
Rst	Input	1 bit	System asynchronous reset
LFSR_out_8	Output	8 bits	The output from the LFSR of lower generations.

○ LFSR_8 MUX



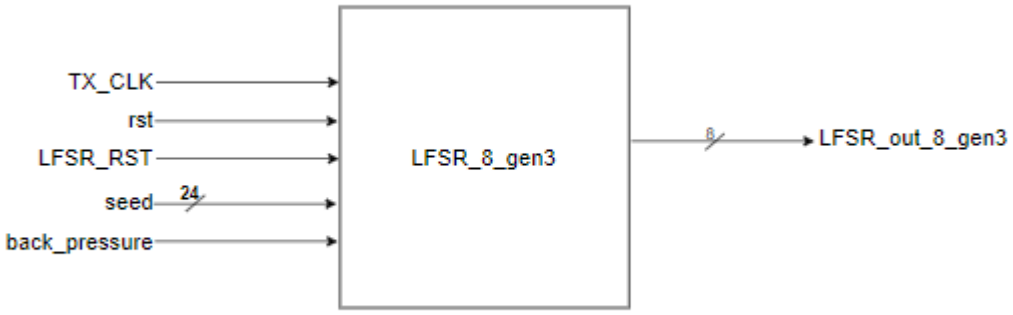
- **Description**

  - The output for the MUX is whether the original data without scrambling or the xoring of the LFSR_8 output with the incoming data. The selection is based on the d_K signal which specifies if the symbol id data or control symbol. If it's a control symbol, it bypasses scrambling. Otherwise, the symbol is scrambled.

- **Ports**

Name	Direction	Size	Description
LFSR_out_8	Input	8 bits	The output from the LFSR of lower generations.
d_K	Input	1 bit	Indicates whether the symbol is data character or control character. 0: data character 1: (K) control character
Sc_Data	Input	8 bits	The input data in each lane after the TX framing.
Data_out_gen_low	Output	8 bits	The scrambled output at lower generations

○ LFSR_8_gen3 block



- **Description**

  - The name LFSR_8_gen3 is created as each lane has an incoming data of 8 bits and these bits are xored with the output of the LFSR which is the advanced 8 bits of the LFSR.
  - LFSR_8_gen3 block is used only for higher generations (Gen 3 and above).

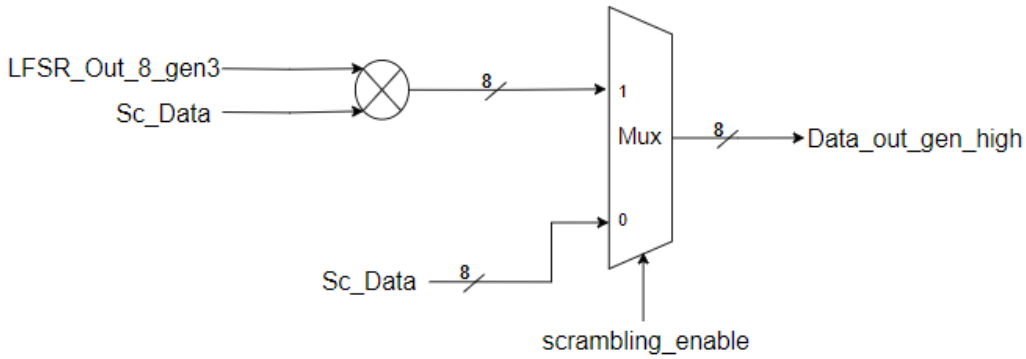
- **Working Mechanism**

  - It's made of a 16-bit Linear Feedback Shift Register (LFSR) with feedback points that implement the following polynomial: $\boldsymbol{G(X) = X^{23} + X^{21} + X^{16} + X^{8} + X^{5} + X^{2} + 1}$
  - Since the input data is 8 bits, it needs 8 output bits to be xored with, however the LFSR outputs 1 bit each clock. The issue can be solved by advancing the LFSR 8 times each cycle, thus the input data is xored each cycle with the 8 outputs from the LFSR.

- **Ports**

Name	Direction	Size	Description
LFSR_RST	Input	1 bit	For resetting the LFSR at the end of EIOS.
TX_CLK	Input	1 bit	Transmitter clock
rst	Input	1 bit	System asynchronous reset
seed	Input	24 bits	The initial values of the LFSR.
back_pressure	Input	1 bit	The back pressure coming from the buffer logic.
LFSR_out_8_gen3	Output	8 bits	The output from the LFSR of higher generations.

- **LFSR_8_gen3 MUX**



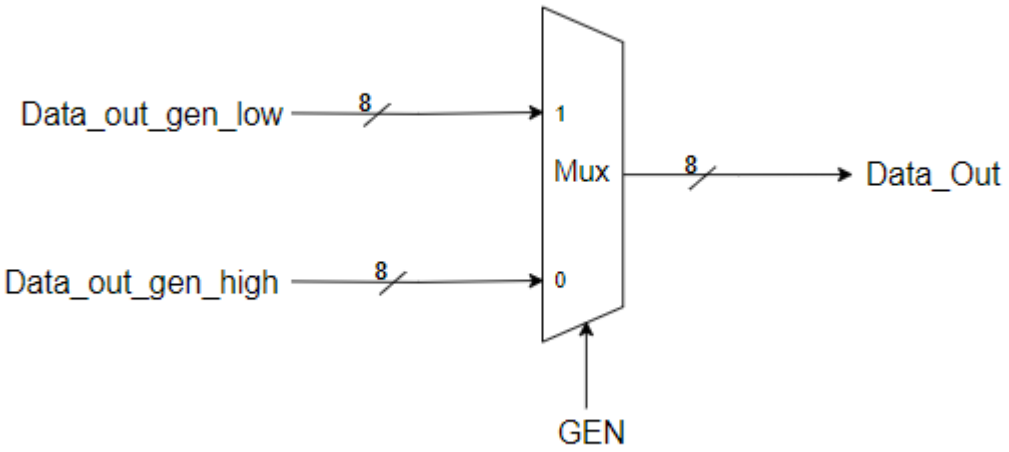
- **Description**

  - The output for the MUX is whether the original data without scrambling or the xoring of the LFSR_8_gen3 output with the incoming data. The selection is based on the scrambling_enable from the control block which specifies if the symbol will bypass scrambling or not based on certain rules.

- **Ports**

Name	Direction	Size	Description
LFSR_out_8_gen3	Input	8 bits	The output from the LFSR of higher generations.
scrambling_enable	Input	1 bit	From the control logic to let the scrambled data or the original data to .
Sc_Data	Input	8 bits	The input data in each lane after the TX framing.
Data_out_gen_high	Output	8 bits	The scrambled output at higher generations

- **GEN MUX**



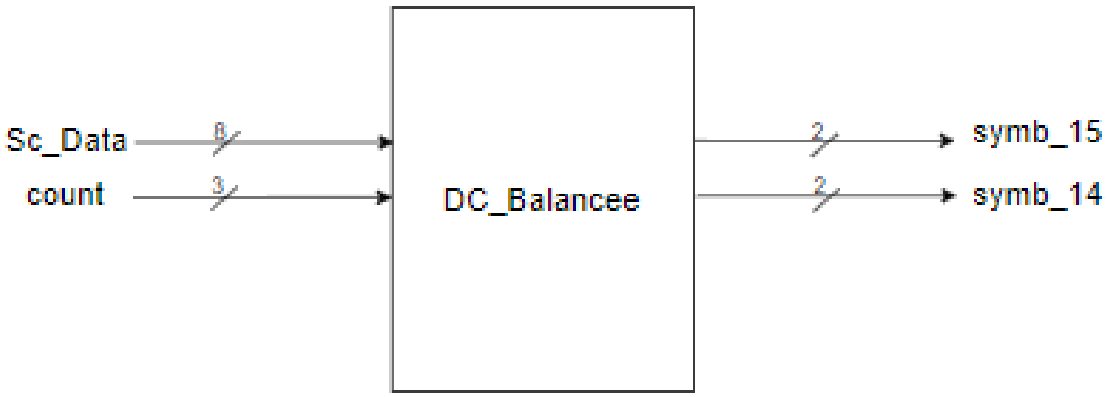
- **Description**

  - The MUX selects the output from the LFSR_8 or LFSR_8_gen3 based on the current working generation of the system.

- **Ports**

Name	Direction	Size	Description
Data_out_gen_low	Input	8 bits	The scrambled data at lower generations
Data_out_gen_high	Input	8 bits	The scrambled data at higher generations
GEN	Input	1 bit	The current generation.
Data_Out	Output	8 bits	The output data from one of the LFSRs whether it's scrambled or not.

- **DC_Balance block**



- **Description**

  - The two problems that 8b/10b encoding automatically addressed—maintaining DC Balance and offering a sufficient transition density—are addressed by altering the scrambling logic for 128b/130b from earlier PCIe generations.
  - DC balance indicates an equal distribution of ones and zeros in the bit stream. This aims to prevent the issue known as “DC wonder,” where the transmission media becomes so heavily biased in favor of one voltage over another due to an excess of ones or zeros that it becomes challenging to flip the signal in the specified period of time.

- **Working Mechanism**

  - For each symbol a counter (dc balance) is used to measure the difference between the number zeros and ones in the symbol. This counter accumulates till it reaches the value 511 and it saturates.
  - The content of symbols 14 and 15 depends of the dc balance value in which:
    - If the running DC Balance value is > 31 at the end of Symbol 11 and more ones have been sent, Symbol 14 = 20h and Symbol 15 = 08h. If more zeroes have been sent, Symbol 14 = DFh and Symbol 15 = F7h.
    - If the running DC Balance value is > 15, Symbol 14 = the normal scrambled TS1 or TS2 identifier, while Symbol 15 = 08h to reduce the number of ones, or F7h to reduce the number of zeroes in the DC Balance count.

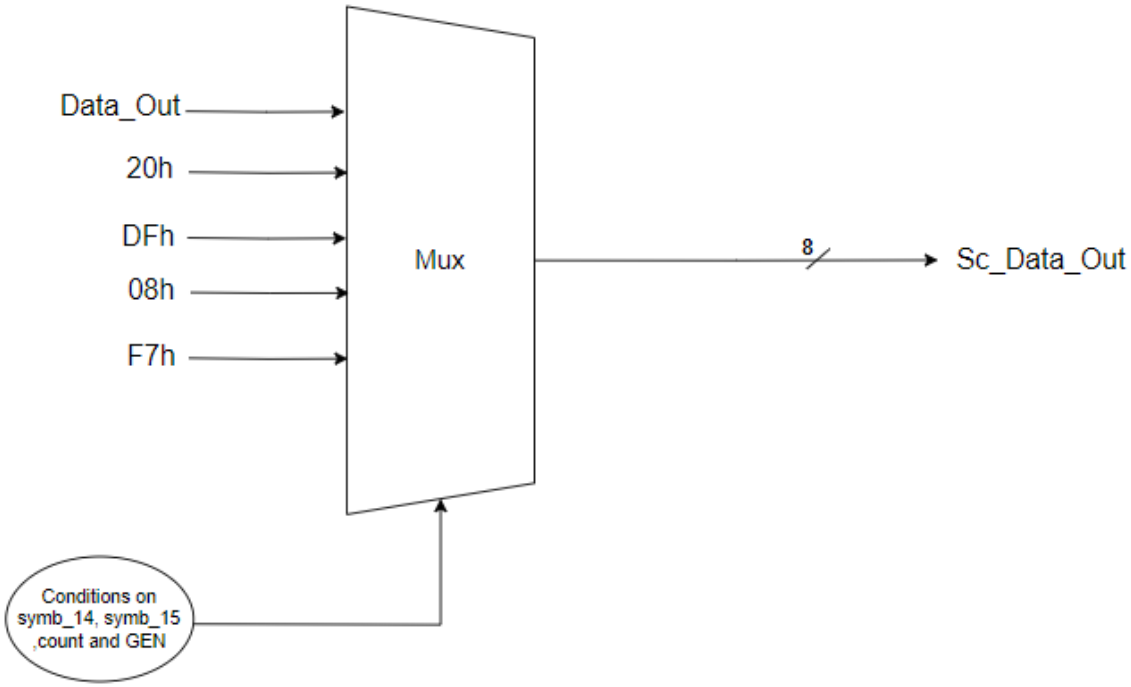
- Otherwise, the normal TS1 or TS2 identifier Symbols will be sent.

- Since symbol 14 can take 2 values only, symb_14 width is 2 bits. 2'b00 means the scrambled output is 20h while 2'b01 means the scrambled output is DFh. Otherwise, the output is the scrambled data.
- Since symbol 15 can take 2 values only, symb_15 width is 2 bits. 2'b00 means the scrambled output is 08h while 2'b01 means the scrambled output is F7h. Otherwise, the output is the scrambled data.

- **Ports**

Name	Direction	Size	Description
Sc_Data	Input	8 bits	The input data in each lane after the TX framing.
count	Input	3 bits	Indicate the current symbol number in the block.
Symb_14	Output	2 bits	Indicates that if the dc value > 31 at the end of symbol 11, symbol 14 should take one of these values: 2'b00: 20h 2'b01: DFh Otherwise, symbol 14 is scrambled.
Symb_15	Output	2 bits	Indicates that if the dc value>15 or the dc value > 31 at the end of symbol 11 , symbol 15 should take one of these values: 2'b00: 08h 2'b01: F7h Otherwise, symbol 15 is scrambled.

- **Sc_Data_Out MUX**



- **Description**

  - The MUX selects the output data from one of the LFSRs or a specified value according to the value of the dc balance block.

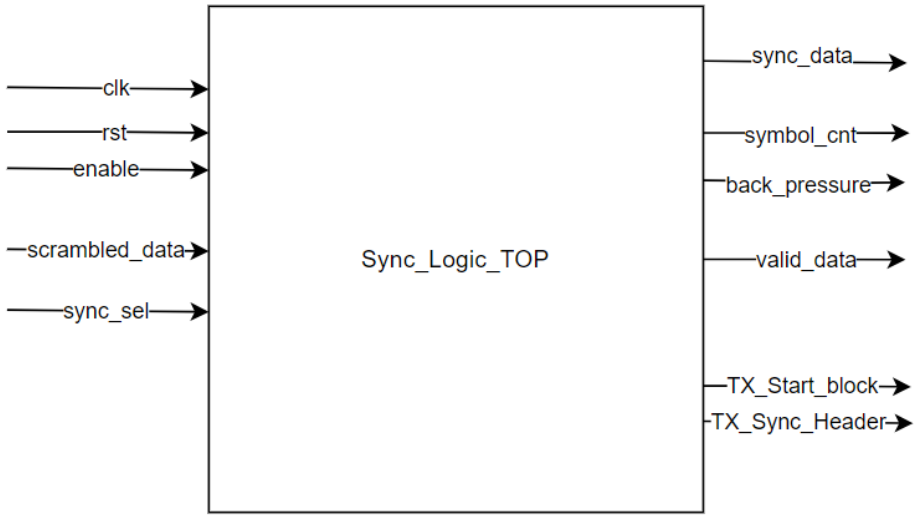
- **Working Mechanism**

  - The condition that specifies the output is simply done by comparing the current count (symbol count) if its value is 14 or 15 and the output from the dc balance block is 00 or 01 in symb_14 or symb_15 signals one of the declared values is forced on the output.
  - For instance, if symb_14 = 2'b00 and count = 4'b1110 the output must be 20h while if symb_15 = 2'b00 and count = 4'b1111 the output must be 08h.
  - The forcing of symbol 14 and 15 is done only at higher generations to maintain the dc balance.
  - Otherwise, the output data is the scrambled data from one of the LFSRs.

- **Ports**

Name	Direction	Size	Description
GEN	Input	1 bit	The current generation.
count	Input	3 bits	Indicate the current symbol number in the block.
Symb_14	Input	2 bits	Indicates that if the dc value > 31 at the end of symbol 11, symbol 14 should take one of these values: 2'b00: 20h 2'b01: DFh Otherwise, symbol 14 is scrambled.
Symb_15	Input	2 bits	Indicates that if the dc value>15 or the dc value > 31 at the end of symbol 11 , symbol 15 should take one of these values: 2'b00: 08h 2'b01: F7h Otherwise, symbol 15 is scrambled.
Data_out	Input	8 bits	The output data from one of the LFSRs whether it's scrambled or not.
Sc_Data_Out	Output	8 bits	The final output of the scrambler.

1.5. Sync Logic



- **Description**

  - This block is responsible for incorporating the sync header, which consists of "10" for ordered sets and "01" for data. Its purpose is to encode the data using the 128b/130b encoding scheme designed for GEN 3 or higher.

- **Working Mechanism**

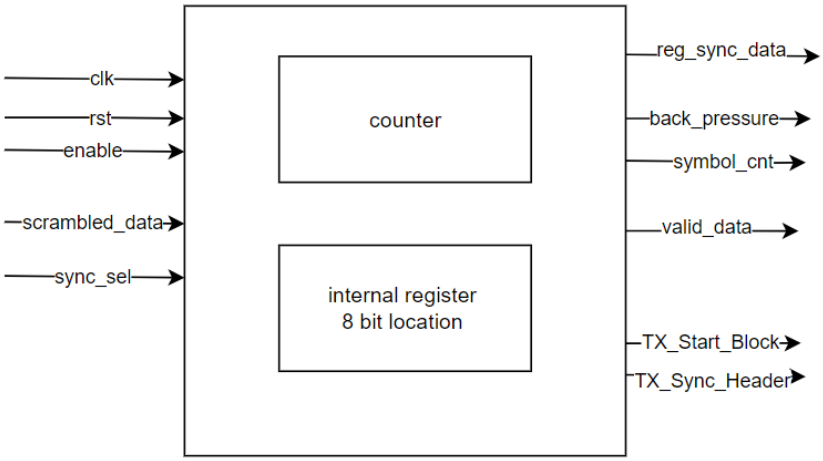
  - The data received from the scrambler is fed into this block. And at the start of each new block (a block is 16 bytes), this logic is responsible for appending the sync header and buffering the remaining bits. (e.g. **scrambled_data**:8'b 10101000 and **sync_sel**: 1 (ordered_set), **sync_data**: 8'b10100010, and the remaining 2'b10 are registered for next transmission).
  - After 3 blocks, and at the beginning of the fourth, there will be 6 bits registered from last transmission and there will be a need to insert the sync header, hence the back_pressure signal is asserted to halt the logic before wards from outputting a new symbol when interacting with the SERDES Architecture, and the PIPE Architecture, additionally if the interface is set to PIPE, the valid_data signal will be de-asserted for that cycle.
  - The enable signal serves to enable the block whenever the negotiated speed is 8GT/s or higher.

- The **sync_sel** signal defines whether the incoming data is data or ordered set to add the correct sync header accordingly.
- TX_Start_Block is asserted with symbol zero in the block in PIPE Architecture.
- TX_Sync_Header determines the value of the sync_header in PIPE Architecture.

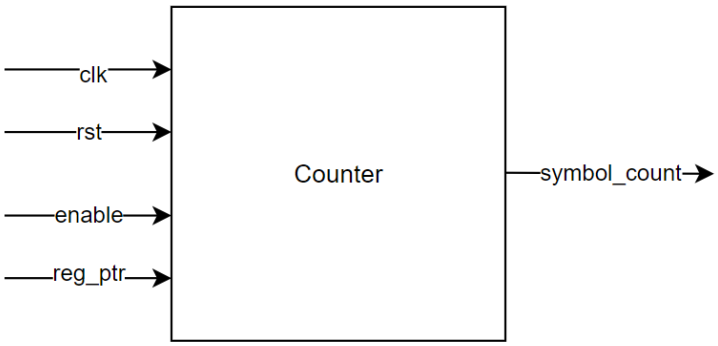
- **Ports**

Name	Direction	Size	Description
Clk	Input	1 bit	The transmitter clock
Rst	Input	1 bit	Asynchronous reset
scrambled_data	Input	8 bits	Input Data from scrambler.
sync_sel	Input	1 bit	Input from Tx_Framing, to know which type of sync headers is to be inserted. A value of 0 means a data block (meaning 01 sync header should be inserted) A value of 1 means an OS block (meaning 10 should be inserted)
Enable	Input	1 bit	Asserted when operating at high data rate (32GT/s).
sync_data	Output	8 bits	Output DATA to be sent to the PIPE.
symbol_cnt	Output	4 bits	Indicate the current symbol count in the block.
back_pressure	Output	1 bit	Input to almost all the logic behind to stop progressing to allow sending the accumulated bits.
valid_data	Output	1 bit	Indicate that the current data on the bus is valid, asserted once the block is enabled. Asserted always in SERDES Architecture, while it gets de-asserted one cycle in PIPE Architecture, when the back_pressure is asserted.
TX_Start_Block	Output	1 bit	Indicate the Start of Block in PIPE Architecture
TX_Sync_Header	Output	2 bits	Indicate the value of the Sync header in PIPE Architecture

- **Implementation**
  - **Sync Logic TOP**



  - **Counter**



- **Description**

  - Up counter that keeps track of the current symbol count

- **Working Mechanism**

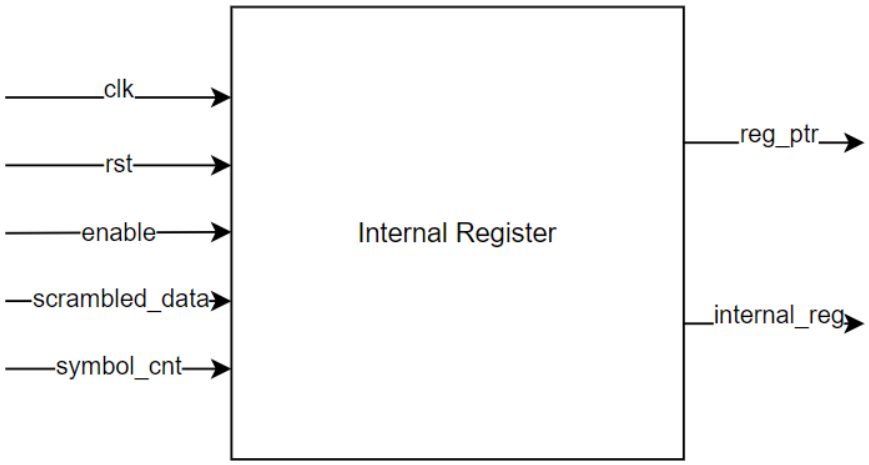
  - The counter starts to count with the **enable** signal asserted and reinitializes each 16 counts, as the block consists of 16 symbols, and whenever the count is zero, the sync header is added.
  - when the **reg_ptr** is six ,a set signal is asserted so that, the next count is one, because the registered six bits will be sent with the

sync header, and symbol zero will be sent the next cycle, and the value of symbol_cnt will hold the value one as the count will not increment due to the backpressure asserted.

- **Ports**

Name	Direction	Size	Description
clk	Input	1 bit	The transmitter clock
rst	Input	1 bit	Asynchronous reset
enable	Input	1 bit	Asserted when operating at 8GT/S or higher.
reg_ptr	Input	3 bits	Indicate the number of the valid bits in the internal register (8 bits location).
symbol_cnt	Output	4 bits	Indicate the current symbol count in the block.

○ Internal Register



- **Description**

- Internal register that serves to buffer the incoming bits not sent due to the addition of sync header to be transmitted the next cycle.

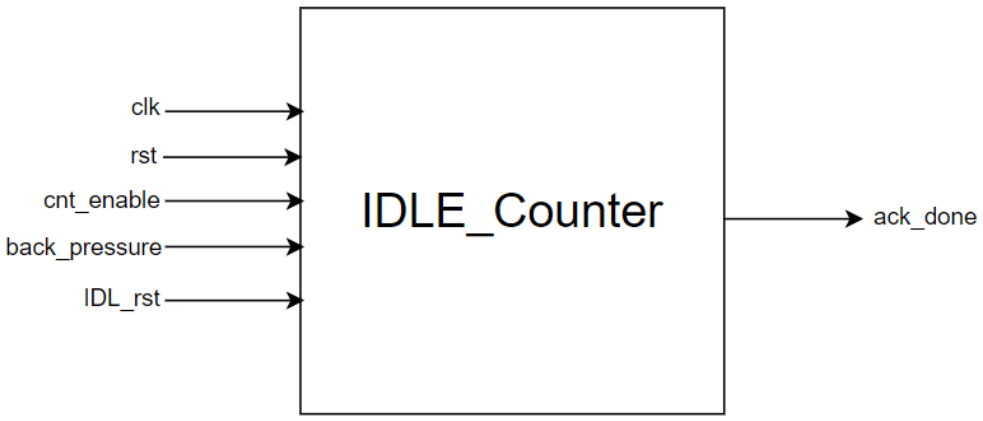
- **Working Mechanism**

  - The scrambled data is fed to the block and the **symbol_cnt** along with the current **reg_ptr** are checked: if the symbol count is zero, that means the need to add the sync header, which in turns increment the **reg_ptr**, and keep the required number of bits buffered.
  - At the beginning the **reg_ptr** is initialized to zero and the **symbol_cnt** is also zero; with the first symbol received the sync header is added and the **reg_ptr** is incremented, and the 2 bits of remaining data are assigned to the **internal_reg.**
  - At the start of the second block, the sync header is added to the 2 bits buffered from last transmission, and 4 bits of the incoming symbol will be taken and the other 4 will be assigned to the **internal_reg** and the **reg_ptr** will be incremented.
  - At the start of the third block, the sync header is added to the 4 bits buffered from last transmission, and 2 bits of the incoming symbol will be taken and the other 6 will be assigned to the **internal_reg** and the **reg_ptr** will be incremented.
  - At the start of the fourth block, the **reg_ptr** will be six and the **symbol_cnt** will be zero, which indicates the insertion of the bits of sync header, hence the 8 bits of the incoming symbol will be buffered and the **reg_ptr** is incremented. The **back_pressure** is also assetered to disable the TX logic from releasing another symbol as for the next cycle the buffered symbol will be outputted.

- **Ports**

Name	Direction	Size	Description
clk	Input	1 bit	The transmitter clock
rst	Input	1 bit	Asynchronous reset
scrambled_data	Input	8 bits	Input Data from scrambler.
enable	Input	1 bit	Asserted when operating at 8GT/S or higher.
symbol_cnt	Input	4 bits	Indicate the current symbol count in the block.
reg_sync_data	Output	8 bits	Output DATA to be sent to the PIPE.
reg_ptr	Output	3 bits	Indicate the number of the valid bits in the internal register (8 bits location).
internal_reg	Output	8 bits	the bits buffered from last symbol transmitted

1.6. IDLE Counter



- **Description**

- The IDLE_Counter module serves to calculate the total number of idles sent and feed the LTSSM with ack_done indicating that the sent idles are 16.

- **Working Mechanism**

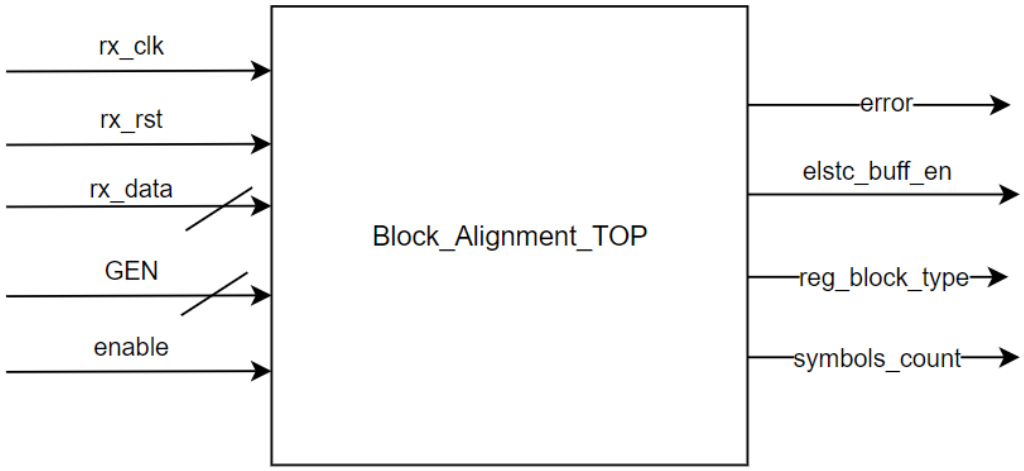
- The counter resets if there is a global rst signal or there is an IDL_rst signal coming from the LTSSM once it enters Configuration_IDLE substate.
- Otherwise, the counter is enabled by the LTSSM to start counting the sent IDLE until the count reaches 16

- **Ports**

Name	Direction	Size	Description
clk	Input	1 bit	TX Clock
rst	Input	1 bit	Global reset signal
cnt_enable	Input	1 bit	Enables the counter to move forward by 1.
back_pressure	Input	1 bit	The back pressure coming from the buffer logic.
IDL_rst	Input	1 bit	Signal from the LTSSM to reset the Idle count to 0.
ack_done	Output	1 bit	When count reaches 16 indicating an acknowledgment for sending 16 idles.

2. Receiver Blocks

2.1. Block Alignment



- **Description**

  - This block serves to determine the block boundary by monitoring for some patterns that are recognizable by the receiver such as EIEOS(electrical idle exit ordered set) and SOS (SKP ordered set).
  - It also determines the block type whether it is data or ordered set and keeps track of the symbol count in the block.

- **Working Mechanism**

  - The BA_FSM makes the transition between the States according to the checks made on the symbol given, with the bit number and the symbol number produced by the BA_counters module.
  - The Block alignment aims to find the start of the block to be able to determine the block type, whether it is data or ordered sets, and remove the sync header, to enable the elastic buffer to register the symbol only with its type and count.

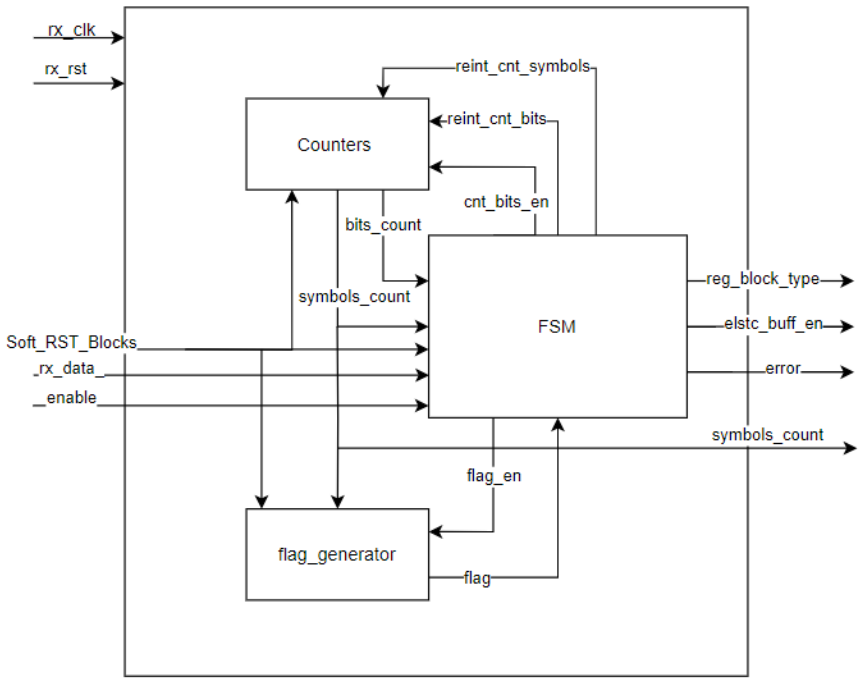
- **Ports**

Name	Direction	Size	Description
rx_clk	Input	1 bit	The receiver clock (bit level clock)

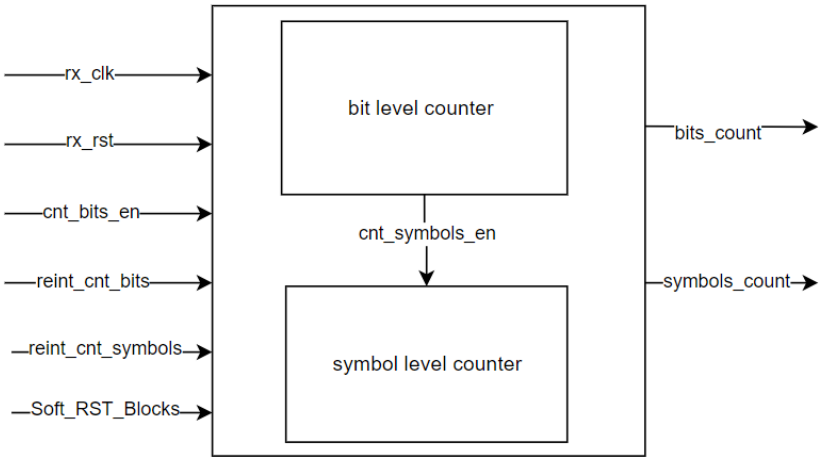
rx_rst	Input	1 bit	Asynchronous reset
rx_data	Input	8 bits	Input Data from the de-serializer in the pipe.
enable	Input	1 bit	Asserted when operating at 8GT/S or higher.
Soft_RST_Blocks	Input	1 bit	Reset the state of the block alignment back to Unaligned State when the recovery process is initiated by the LTSSM
rst_BA	Input	1 bit	Reset the error flag asserted by the block alignment when an undefined sync header is detected.
elstc_buff_en	Output	1 bit	Act as the write enable to the elastic buffer
symbols_count	Output	4 bits	Indicates the symbol count in the block.
reg_block_type	Output	1 bit	A value of 0 means a data block. A value of 1 means an OS block .
Error	Output	1 bit	Indicate the occurrence of an error: (undefined sync or incorrect SKP symbol) received while in locked phase

- **Implementation**

  - **Block Alignment TOP**



Counters



- **Description**

  - This block includes two Up counters: the vit level increments each **rx_clk**, and the symbol level increments each eight counts of the bit level counter.

- **Working Mechanism**

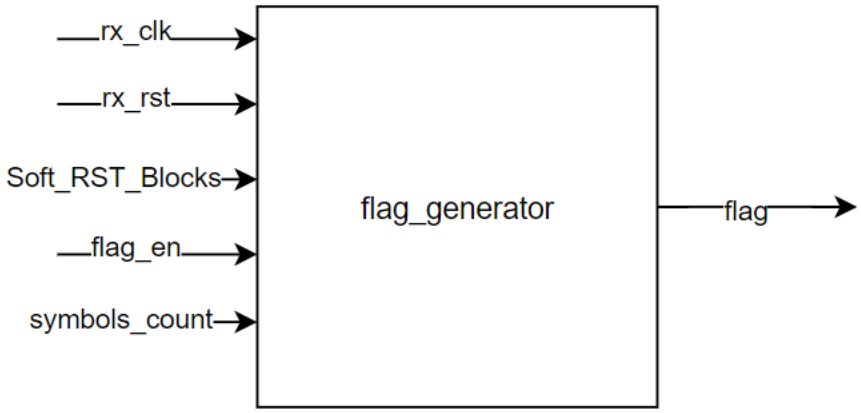
  - Once the **cnt_bits_en** (input from the FSM when the beginning of a symbol is detected) is asserted, the bit level counter starts to increment till it reaches 3'b111, and then reinitializes to zero.
  - When the **bits_count** reaches 3'b111 it asserts the **cnt_symbols_en** signal to increment the symbols counted by one.
  - The symbol counter increments each time the **cnt_symbols_en** is asserted, and finally reinitializes once the **symbol_count** reaches 4'b1111.
  - The **reint_cnt_bits** signal serves to return the counter to zero again, it is input from the FSM, and is asserted after passing the 2 bits of the sync header to start counting the bits of the symbol again.
  - The **reint_cnt_symbols** signal is needed to be asserted in case after the reception of multiple symbols of a block, an incorrect symbol is detected, and there is a need to redefine the block boundary and restart the counting.
  - When **Soft_RST_Blocks** signal is asserted, the bits and symbols counters are rested to zero, and do not increment, until directed by the BA_FSM module when the first symbol of the BA is seen correctly.

- **Ports**

Name	Direction	Size	Description
rx_clk	Input	1 bit	The receiver clock (bit level clock)
rx_rst	Input	1 bit	Asynchronous reset
cnt_bits_en	Input	1 bit	Input from FSM to enable the bit counter
reint_cnt_bits	Input	1 bit	Input from FSM to reinitializes the bit counter to zero
reint_cnt_symbols	Input	1 bit	Input from FSM to reinitializes the symbol counter to zero

Soft_RST_Blocks	Input	1 bit	Reset the value of the counters when the transition is made to the recovery state in the LTSSM.
bits_count	Output	3 bits	Indicate the counted bits of the symbol
symbols_count	Output	4 bits	Indicates the symbol count in the block.

○ Flag Generator



- ## Description

  - This block is responsible for the generation of the flag that helps the FSM keep track of the EIEOS.

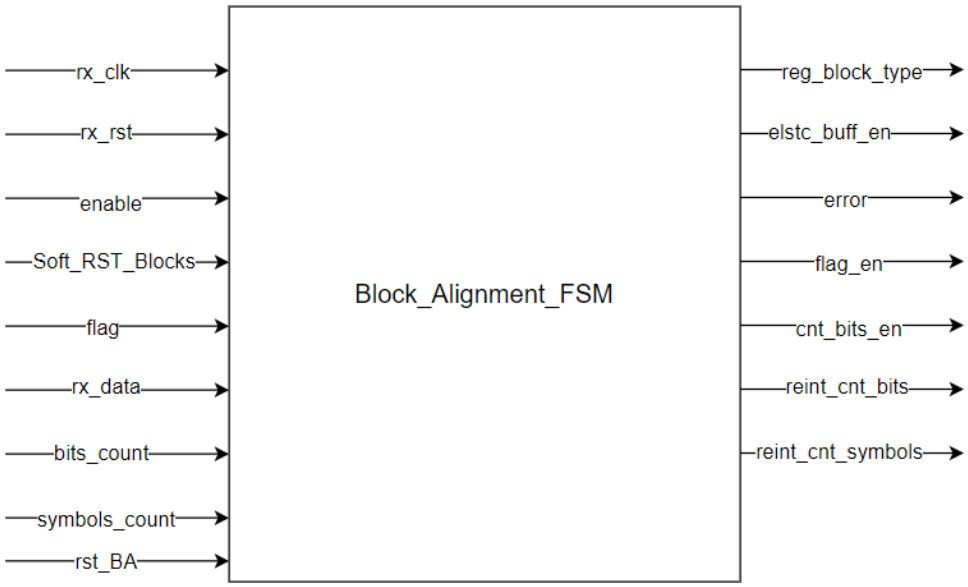
- ## Working Mechanism

  - The FSM asserts the **flag_en** signal once the monitoring EIEOS state is entered, which enables this block and start outputting the right value of the flag.
  - The EIEOS consists of alternating zeros and ones according to the **symbols_count**, the alternation frequency also differs according to the generation.
  - The value of the flag is reset back to zero when the **Soft_RST_Blocks** signal is asserted indicating the transition of the LTSSM to the recovery state.

- **Ports**

Name	Direction	Size	Description
rx_clk	Input	1 bit	The receiver clock (bit level clock)
rx_rst	Input	1 bit	Asynchronous reset
flag_en	Input	1 bit	Asserted by the BA_FSM to be able to change the value of the flag according to the state of block alignment.
symbols_count	Input	4 bits	Indicates the symbol count in the block.
Soft_RST_Blocks	Input	1 bit	Reset the flag value to zero when the transition is made to the recovery state.
Flag	Output	1 bit	Indicate the value of the EIEOS that should be received.

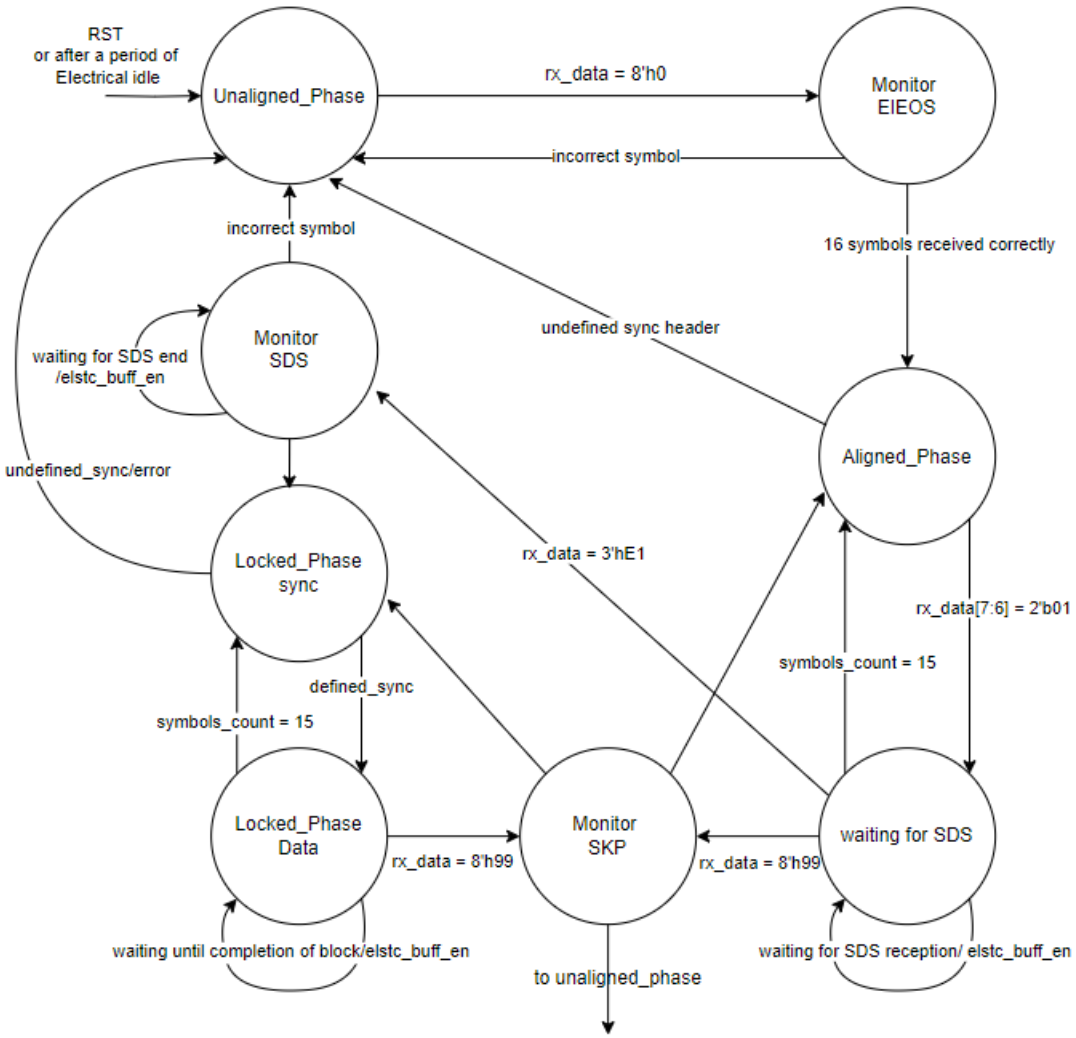
- **Block Alignment FSM**



- **Description**

  - The block alignment FSM is responsible for monitoring different patterns at different instances to define the block boundary.

- **Working Mechanism**



- **Unaligned_phase:** this is the state where the block starts after a reset or after being in electrical idle, the incoming bits are monitored, until a symbol of all zeros (8'h0) is seen, then the next state is monitor EIEOS.
- **Monitor EIEOS**: In this state, the pattern of the EIEOS is monitored, if the 16 symbols are received correctly the next state is **Aligned_Phase**, if one of the symbols is wrong, the next state is **Unaligned_Phase.**
- **Aligned_Phase:** In this state the 2 bits of sync header are checked, if they are "10" (ordered set sync header), the next state is **Unaligned_Phase**, if the sync header is correct the next state is **waiting for SDS**.
- **Waiting for SDS:** In this state, the **elstc_buff_en** is asserted whenever **bits_count** is 3'b111 to buffer the received symbol. While

waiting for SDS to be received, and after the end of each block, the transition is made to the **Aligned_Phase** to check for the two bits of sync_header, if **symbols_count** is zero, the received symbol is checked: if it is 8'h99, the next state is **Monitor SKP**, and if it is 8'E1, the next state is **Monitor SDS.**

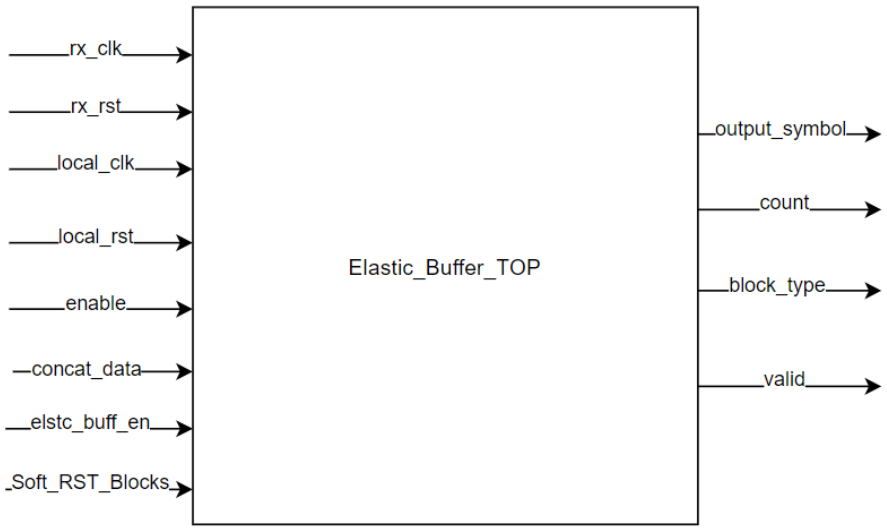
- **Monitor SDS:** In this state the pattern of the SDS is monitored, if the 16 symbols are received correctly, the next state is **locked_phase_sync,** and that indicate the end of training successfully ; and if one of the symbols is wrong the next state is **Unaligned_Phase.**
- **Locked_Phase_sync:** In this state, the 2 bits of sync header are checked, and if they are a defined sync the next state is **Locked_Phase_Data**, but if the sync header is undefined, the next state is **Unaligned_Phase** and the error signal is asserted so that the physical layer initiates the recovery process.
- **Locked_Phase_Data:** In this state, the data is received, and whenever the **symbols_count** is 4'b1111 the next state is **Locked_Phase_sync**, where the 2 bits of sync header get checked for. If the **symbols_count** is 4'b0 and the received symbol is 8'hAA, the next state is **Monitor SKP.**
- **Monitor SKP:** In this state, the pattern of the SKP ordered set is monitored and If all the symbols are received correctly, the next state is the state that made the transition to **Monitor SKP** whether it was **waiting for SDS** or **Locked_Phase_Data**. In case one of the symbols is wrong the next state is **Unaligned_Phase** and depending on the state that the transition was made from the **error signal** will be asserted or not. If the transition was made from **locked_Phase_data** the **error** signal will be asserted, otherwise it will no

- **Ports**

Name	Direction	Size	Description
rx_clk	Input	1 bit	The receiver clock (bit level clock)
rx_rst	Input	1 bit	Asynchronous reset
rx_data	Input	8 bits	Input Data from the de-serializer in the pipe.
enable	Input	1 bit	Asserted when operating at 8GT/S or higher.
Flag	Output	1 bit	Input from the flag generator block, Indicate the value of the EIEOS that should be received. 1'b1: all ones symbol of EIEOS

			1’b0: all zeros symbol of EIEOS
bits_count	Input	3 bits	Input from the counters block, Indicate the counted bits of the symbol
symbols_count	Input	4 bits	Input from the counters block, Indicates the symbol count in the block.
reg_block_type	Output	1 bit	One bit output that indicate the type of the block: 1’b1: ordered set 1’b0: data
elstc_buff_en	Output	1 bit	Act as the write enable to the elastic buffer
error	Output	1 bit	Indicate the occurrence of an error: (undefined sync or incorrect SKP symbol) received while in locked phase.
flag_en	Output	1 bit	Enable the flag generator block, while in monitor EIEOS state.
reint_cnt_bits	Input	1 bit	Reinitializes the bit counter to zero.
reint_cnt_symbols	Input	1 bit	Reinitializes the symbol counter to zero.
rst_BA	Input	1 bit	Reset the error flag of the block alignment when the transition to the recovery state is made by the LTSSM
Soft_RST_Blocks	Input	1 bit	Reset the state of the Block Alignment when the recovery process is initiated by the LTSSM.
cnt_bits_en	Output	1 bit	Enables the bit counter once the beginning of a symbol is detected.

2.2. Elastic Buffer



- **Description**

  - Elastic Buffer is implemented to cross the received data from Recovered Clock Domain (that it uses to latch the inbound data) to Local Clock Domain (that it uses to clock all of the internal gates and to transmit data with) as each device has 2 clock domains: (1) The Recovered Clock Domain, and (2) The local Clock Domain.

- **Working Mechanism**

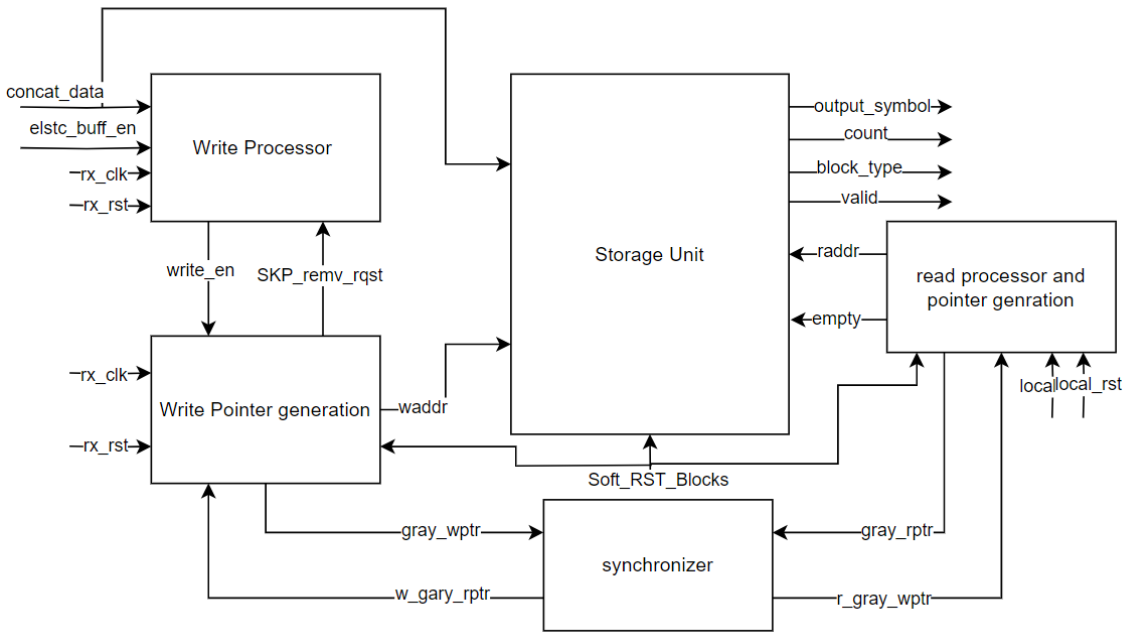
  - We Compensate between the two different clock domains by removing SKP symbols which are sent periodically every 372-375 blocks in GEN 3.0 or higher and we have 2 scenarios that:
  - The Local Clock is faster than the Recovered Clock so, more symbols will be pulled out of the Elastic Buffer than have been deposited into it and the valid signal will be de-asserted to indicate that the buffer is empty .
  - The recovered Clock is faster than the Local Clock so, more symbols will be deposited into the Elastic Buffer than have been pulled out of it and this compensated by removing SKP symbols from the received SKP ordered sets.

- **Ports**

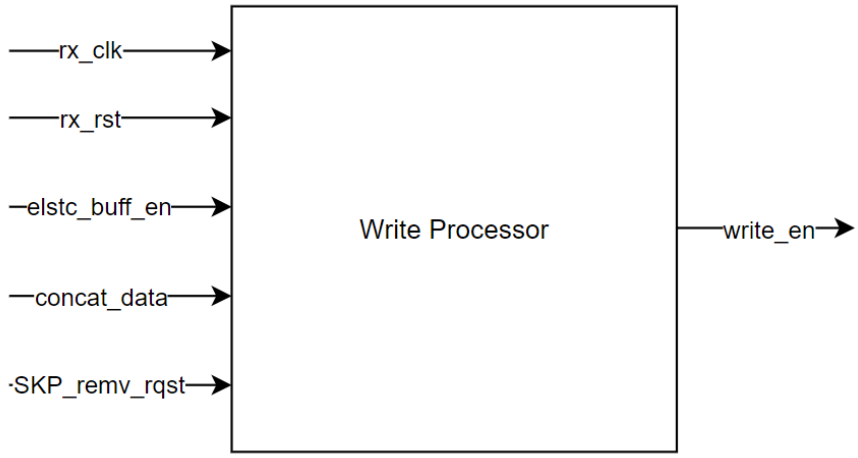
Name	Direction	Size	Description
rx_clk	Input	1 bit	The receiver clock (bit level clock)
rx_rst	Input	1 bit	Asynchronous reset.
local_clk	Input	1 bit	The local clock.
local_rst	Input	1 bit	Asynchronous reset.
enable	Input	1 bit	Asserted when operating at 8GT/S or higher.
elstc_buff_en	Input	1 bit	Act as the write enable to the elastic buffer
concat_data	Input	13 bits	consists of the rx_data (8 bits data coming from the pipe), symbol count (4 bits output of block alignment) and block_type(1 bit output from the block_alignment).
Soft_RST_Blocks	Input	1 bit	Clear the buffer when the recovery process is initiated.
output symbol	Output	8 bits	Output symbol
count	Output	4 bits	Output symbol count.
block_type	Output	1 bit	Output block type
valid	Output	1 bit	Indicate the validity on the bus.

- **Implementation**

  - **Elastic Buffer TOP**



○ Write Processor



- ## Description

  - This block carries out the function of enabling the writing process in the elastic buffer according to the incoming data, the **elstc_buff_en** and the **SKP_remv_rqst** signals.

- ## Working Mechanism

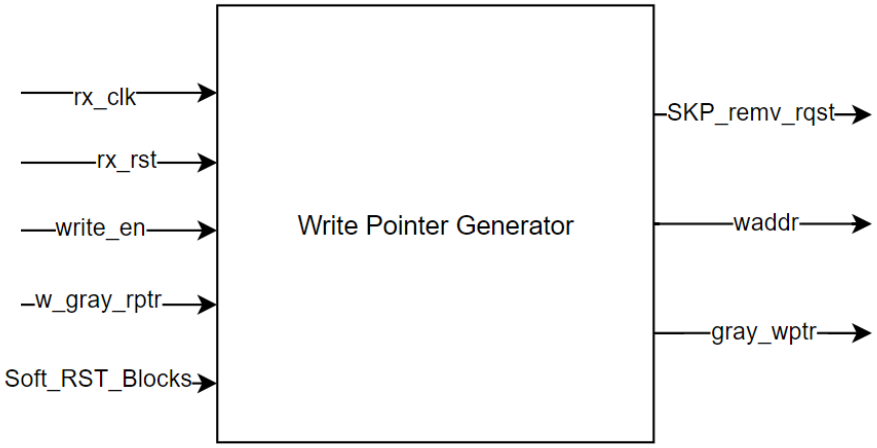
  - The **rx_data**(data coming from pipe), along with **elstc_buff_en** signal, the symbol count, and the **block_type** from the block alignment logic are fed into this block, which checks the incoming data, and asserts or de-asserts the **write_en** signal.
  - If the incoming symbol is SKP and the **SKP_remv_rqst** is asserted, and less than eight SKPs has already been removed from this SOS(SKP ordered set), the **write_en** will not be asserted, and the SKP symbol will not be written into the buffer, otherwise the **write_en** signal is asserted whenever the **elstc_buff_en** is asserted.

- ## Ports

Name	Direction	Size	Description
rx_clk	Input	1 bit	The receiver clock (bit level clock)
rx_rst	Input	1 bit	Asynchronous reset
elstc_buff_en	Input	1 bit	Indicate the validity of the current symbol coming from the pipe.
concat_data	Input	13 bits	consists of the rx_data (8 bits data coming from the pipe),

			symbol count (4 bits output of block alignment) and block_type(1 bit output from the block_alignment).
SKP_remv_rqst	Input	1 bit	Input from write pointer generation block and is asserted whenever the difference between the read and write enable exceeds certain threshold, to be able to delete incoming SKP and compensate for the clock difference.
write_en	Output	1 bit	enable the writing process in the elastic buffer.

- **Write Pointer Generation**



- **Description**

  - This section is responsible for generating the write pointer, which is used to track the state of the buffer. Additionally, it handles the generation of the write address.

- **Working Mechanism**

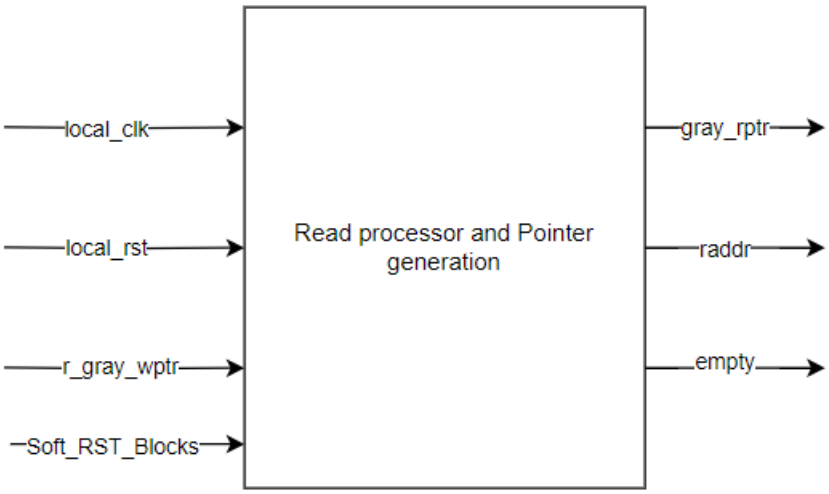
  - Once the **write_en** is asserted, the **wptr** and the **waddr** are incremented to points to the new location.

- The **wptr** is gray encoded to **gray_wptr** and synchronized to the other domain.
- Meanwhile the synchronized gray encoded read pointer **w_gray_rptr** is fed as input to the block and translated to its binary version to calculate the difference between the read and write pointers and determine whether there is a need to remove SKP from the buffer, and hence assert the **SKP_remv_rqst.**
- For the difference calculation and due to the 2 clks difference due the synchronization, the **wptr** is delayed by two clocks and then the difference is calculated to mitigate the effect of synchronization.
- Whenever the **Soft_RST_Blocks** signal is asserted, the write pointer is set back to zero to clear the buffer.

- **Ports**

Name	Direction	Size	Description
rx_clk	Input	1 bit	The receiver clock (bit level clock)
rx_rst	Input	1 bit	Asynchronous reset
write_en	Input	1 bit	enable the writing process of the data.
w_gray_rptr	Input	4 bits	gray encoded read pointer synchronized to the write domain
Soft_RST_Blocks	Input	1 bit	Reset the write pointer back to zero, in order to clear the buffer.
SKP_remv_rqst	Output	1 bit	Asserted whenever the difference between the read and write enable exceeds certain threshold, to be able to delete incoming SKP and compensate for the clock difference.
waddr	Output	3 bits	Indicate the address where the data will be written.
gray_wptr	Output	4 bits	gray encoded write pointer

○ Read processor and pointer generation



- ## Description

  - This block is responsible for the read pointer calculation, as well as the read address.

- ## Working Mechanism

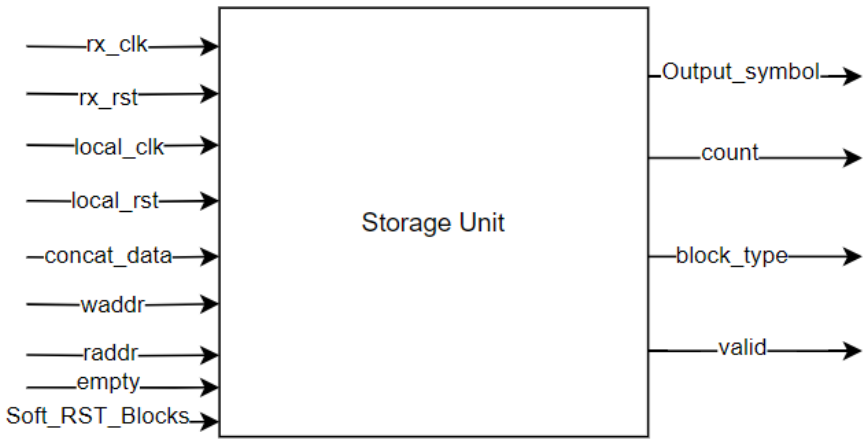
  - Once the block is enabled when the negotiated speed is 8GT/s or higher, the read and write pointers are checked to determine whether the buffer is empty or has data.
  - And once the buffer is detected to not being empty, the **raddr** and the **rptr** are incremented.
  - And the **rptr** is gray encoded to synchronize it to the recovered clock (writing) domain.
  - Whenever the **Soft_RST_Blocks** signal is asserted, the read pointer is set back to zero in order to clear the buffer.

- ## Ports

Name	Direction	Size	Description
local_clk	Input	1 bit	The local Clock.
local_rst	Input	1 bit	Asynchronous local reset.
r_gray_wptr	Input	1 bit	gray encoded write pointer synchronized to the read domain.

Soft_RST_Blocks	Input	1 bit	Reset the read pointer back to zero, in order to clear the buffer.
gray_rptr	Output	4 bits	gray encoded read pointer.
raddr	Output	1 bit	Indicates the address where the data will be read from.
empty	Output	1 bit	Indicate whether the buffer is empty or not.

○ Storage Unit



- ## Description

  - Represents the actual storage element in the elastic buffer, where the data is written and read from.

- ## Working Mechanism

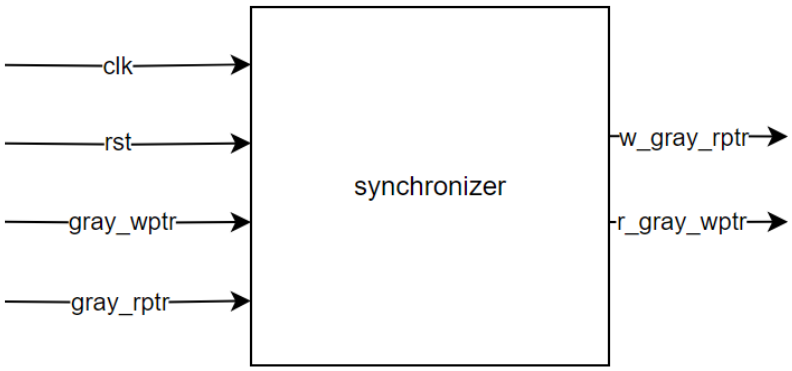
  - With the **write_en** asserted the data is written in the given **waddr.**
  - And whenever the buffer is not empty the data is read out of the buffer each clock (local clock) and the valid signal is asserted.
  - Whenever the **Soft_RST_Blocks** signal is asserted, the buffer is cleared.

- ## Ports

Name	Direction	Size	Description
rx_clk	Input	1 bit	The receiver clock (bit level clock)
rx_rst	Input	1 bit	Asynchronous reset.

local_clk	Input	1 bit	The local clock.
local_rst	Input	1 bit	Asynchronous reset.
waddr	Input	3 bits	Indicate the address where the data will be written.
raddr	Input	1 bit	Indicates the address where the data will be read from.
concat_data	Input	13 bits	consists of the rx_data (8 bits data coming from the pipe), symbol count (4 bits output of block alignment) and block_type(1 bit output from the block_alignment).
Soft_RST_Blocks	Input	1 bit	Clear the buffer when the recovery process is initiated.
output symbol	Output	8 bits	Output symbol
count	Output	4 bits	Output symbol count.
block_type	Output	1 bit	Output block type
valid	Output	1 bit	Indicate the validity on the bus.

- **Synchronizer**



- ## Description

  - Two flip flop synchronizer that serves to synchronize the pointers across the two domains.

- ## Working Mechanism

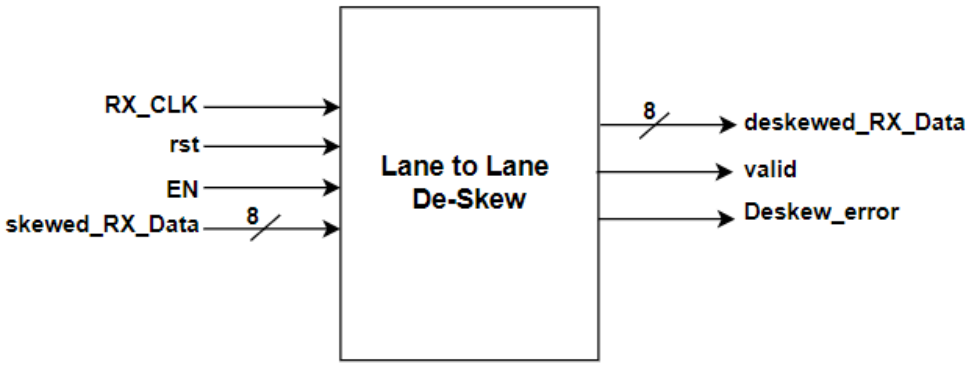
  - There are two instances of this module, one that takes the local clk as the input clock and serves to synchronize the **gray_wptr** to the read (local clock) domain.

- And the other takes the recovered clk as the input clock and serves to synchronize the **gray_rptr** to the writing (recovered clock) domain.

- **Ports**

Name	Direction	Size	Description
Clk	Input	1 bit	The Clock of the domain the pointer is synchronized to .
Rst	Input	1 bit	Asynchronous local reset.
gray_wptr	Input	4 bits	gray encoded write pointer
gray_rptr	Output	4 bits	gray encoded read pointer.
r_gray_wptr	Output	4 bits	gray encoded write pointer synchronized to the read domain.
w_gray_rptr	Output	4 bits	gray encoded read pointer synchronized to the write domain.

2.3. Lane to Lane Deskew



- **Description**

- Lane to Lane De-skew logic is implemented to re-align the phases of the received data effectively as the receiver device may not receive the data on each lane at the exact same time due to the differences between electrical drivers and receivers or due to some other board characteristics.
- De-skewing data is accomplished by simply delaying the faster lanes by the appropriate number of clocks until watching SOS or SDS at

Higher GENs on all lanes. Accordingly, now the receiver can process the data.

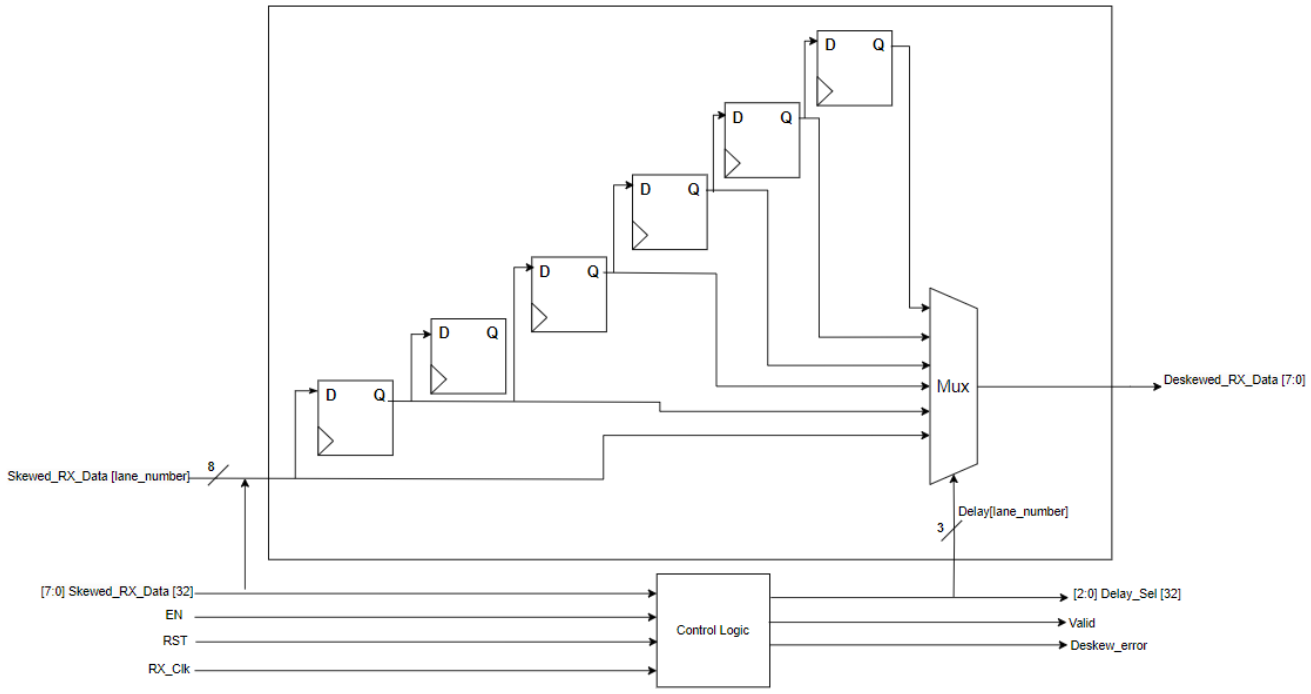
- **Working Mechanism**

  - The control block of the lane-to-lane de-skew will have the full access on all of the lanes and will decide how much delay is needed for each lane based on the time of arrival of the SDS ordered set.
  - The amount of allowed skew that can be corrected at the receiver is 6 clock cycles. (6 Storing Registers)
  - The valid signal in a certain lane is asserted if the selected delay for such lane is less than 6 clock cycles and all of the lanes sees an SDS.
  - The De-skew error in a certain lane is asserted if the selected delay exceeded 6 clock cycles and will inform the LTSSM to take action.

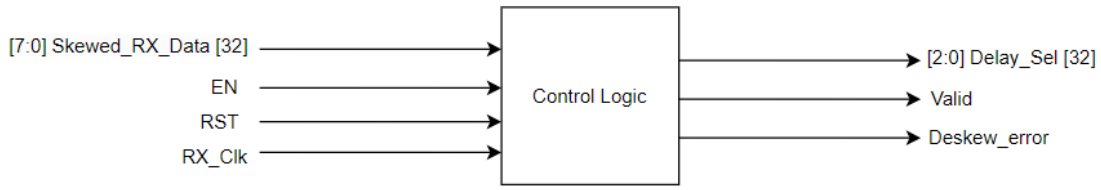
- **Ports**

Name	Direction	Size	Description
skewed_RX_Data	Input	8 bits	Input Data received from the Elastic buffer on one lane
EN	Input	1 bit	Enable from the elastic buffer to allow the logic to take place.
RX_CLK	Input	1 bit	Receiver recovered clock
Rst	Input	1 bit	System asynchronous low reset
deskewed_RX_DATA	Output	8 bits	Output from Lane to Lane De-skew when data is aligned on all lanes and ready to be processed.

- **Implementation**



- **Deskew Control logic**



- **Description**

  - The Deskew control logic selects the appropriate delay for each of the 32 lanes such that the lanes are aligned.

- **Working Mechanism**

  - The control block checks all the lanes for SDS ordered set, if all the lanes have an SDS the delay select for all lanes will be the original skewed data. However, if one lane didn't see an SDS all the other

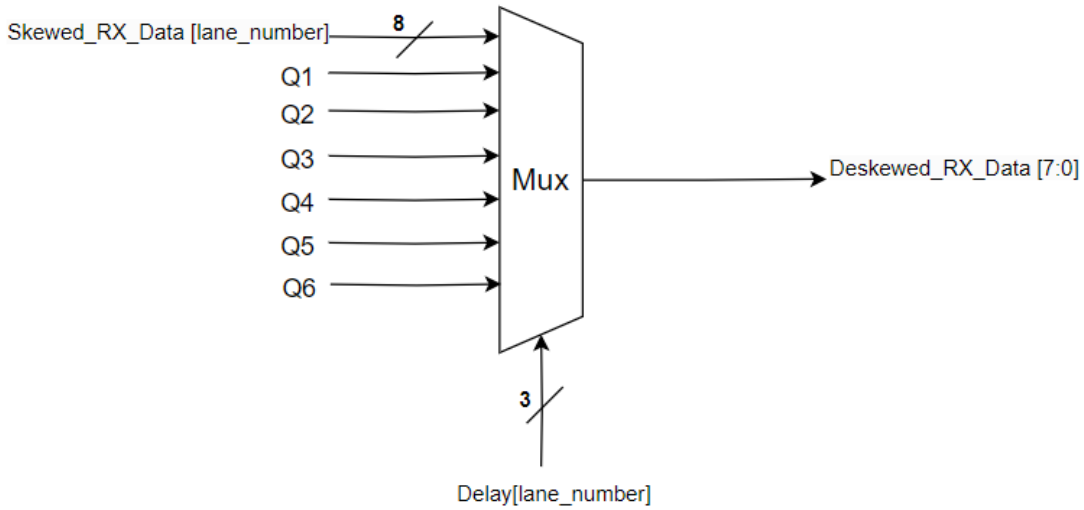
lanes will be delayed one clock cycle and the output of those lanes are taken from the output of the first flip flop while the delayed lane output is taken from the original skewed data.

- For instance, lane 3 SDS came at the third cycle while lane 7 SDS came at the fifth cycle thus the delay_select = 32'h555**2**555**0**55555555555555555555555

- **Ports**

Name	Direction	Size	Description
skewed_RX_Data	Input	8 bits * 32	Input Data received from the Elastic buffer from all lanes.
EN	Input	1 bit	Enable from the elastic buffer to allow the logic to take place.
RX_CLK	Input	1 bit	Receiver recovered clock
delay_select	Internal signal	3 bits	Selects the amount of the delay for a certain lane.
rst	Input	1 bit	System asynchronous low reset
valid	Output	1 bit	Indicates that the data is valid for the descrambler input.
Deskew_error	Output	1 bit	Reports an error to the LTSSM

- **Deskew MUX**



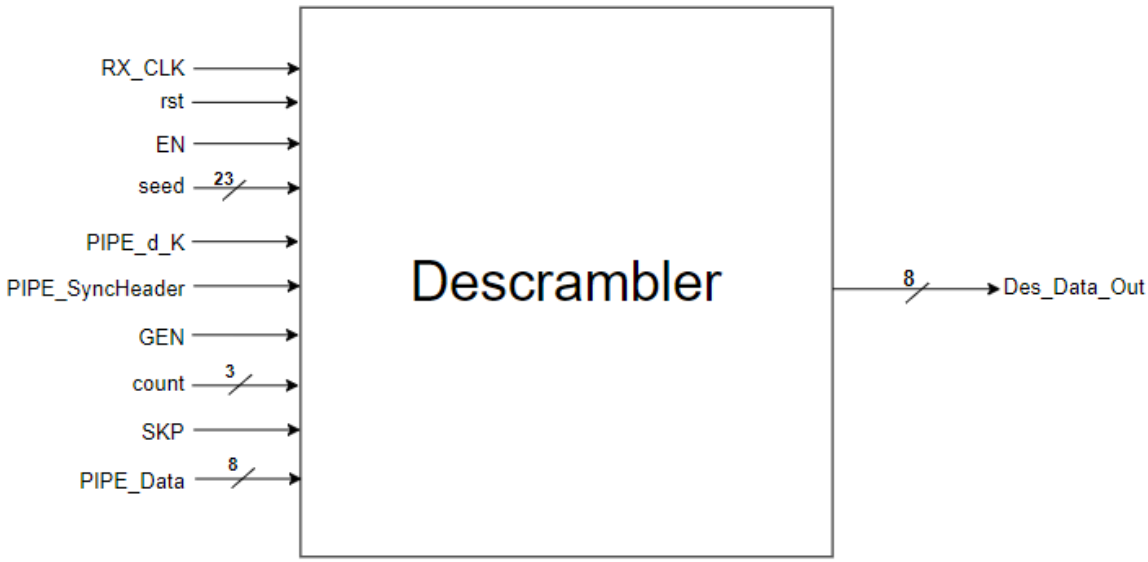
- **Description**

    - The Mux selects the deskewed data from one of 6 registers or from the original skewed data based on the delay_select signal from the control logic on a certain lane .

- **Ports**

Name	Direction	Size	Description
skewed_RX_Data	Input	8 bits	Input Data received from the Elastic buffer on one lane.
delay_select	Input	3 bits	Based on the control logic it selects the output of one of the registers.
Q1	Input	1 bit	The data arising from the first register.
Q2	Input	1 bit	The data arising from the second register.
Q3	Input	1 bit	The data arising from the third register.
Q4	Input	1 bit	The data arising from the fourth register.
Q5	Input	1 bit	The data arising from the fifth register.
Q6	Input	3 bits	The data arising from the sixth register.
Dekewed_RX_Data	Output	8 bits	Output from MUX when data is aligned on all lanes and ready to be processed.

2.4. Descrambler



- **Description**

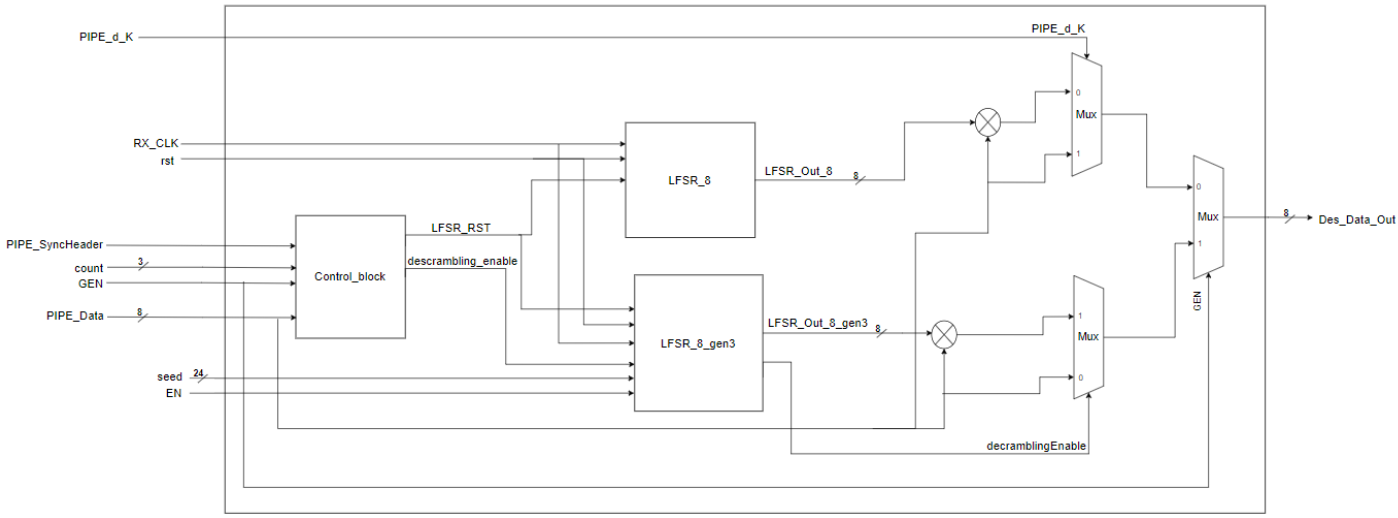
  - Receivers follow exactly the same rules for generating the scrambling polynomial that the Transmitter does and simply XOR the same value to the input data a second time to recover the original information.

- **Ports**

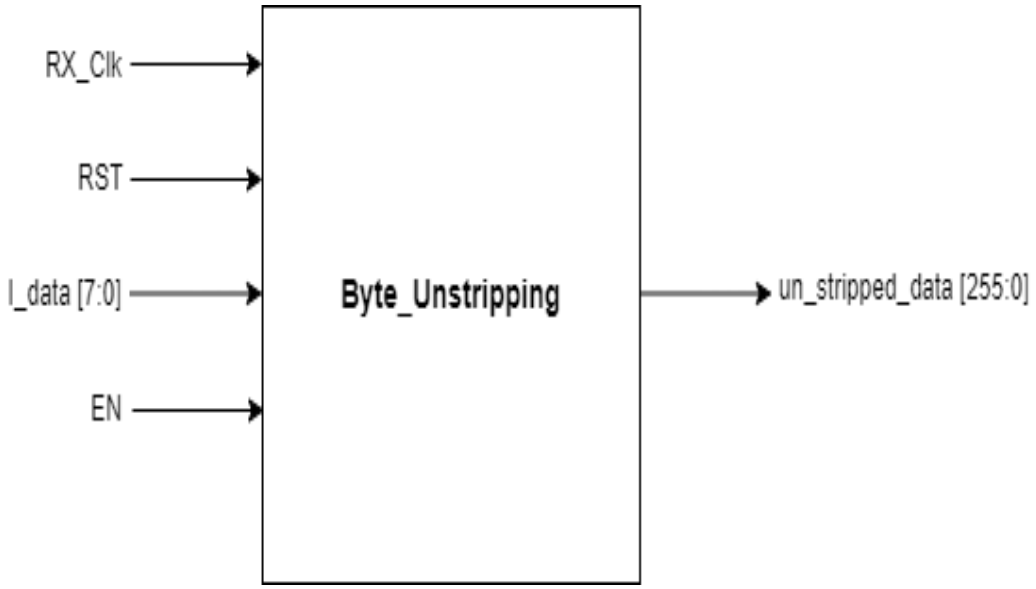
Name	Direction	Size	Description
RX_CLK	Input	1 bit	Receiver recovered clock
Rst	Input	1 bit	System asynchronous reset
PIPE_d_K	Input	1 bit	Indicates whether the symbol coming from the pipe is data character or control character. 0: data character 1: (K) control character
PIPE_SyncHeader	Input	1 bit	Indicate the block type for GEN3: 1'b0: Ordered Set 1'b1: Data It's sent from the Block alignment.
Count	Input	3 bits	Indicate the current symbol number in the block.

			It’s sent from the Block alignment.
PIPE_Data	Input	8 bits	For lower generations it’s sent from the PIPE, however in higher generations it’s sent from the elastic buffer.
GEN	Input	1 bit	The current generation.
seed	Input	24 bits	The initial values of the LFSR coming from the LTSSM.
EN	Input	1 bit	Input from Elastic buffer to enable or disable the block logic.
Dec_Data_Out	Output	8 bits	The descrambler data which is the output of the LFSR xored with the input PIPE_data or the original PIPE_data if it bypassed the LFSR.

- **Implementation**



2.5. Byte unstripping



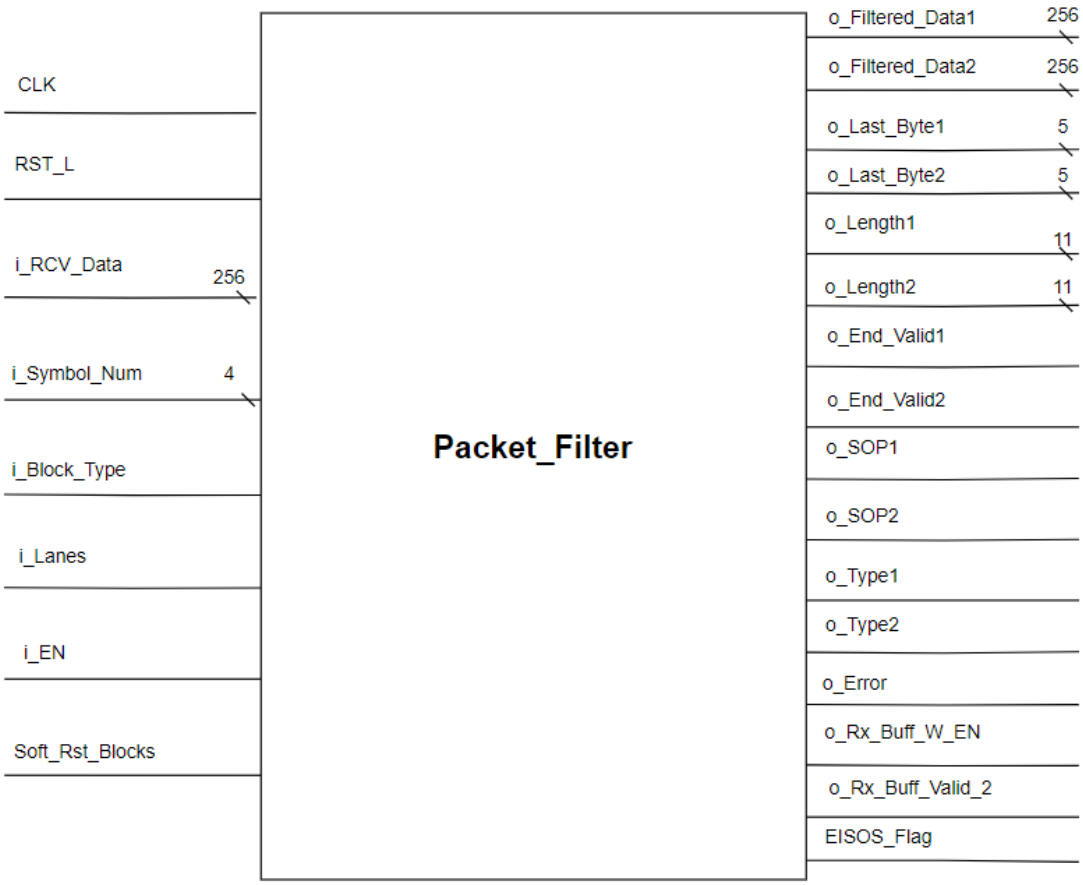
- **Description**

  - Byte un-stripping refers to a technique used to reconstruct data that has been stripped across multiple PCIe lanes.
  - The supported lane widths in our implantation are 1 lane and 32 lanes. Thus, N is either 1 or 32.

- **Ports**

Name	Direction	Size	Description
RX_CLK	input	1 bit	Recover recovered clock
RST	input	1 bit	System asynchronous reset.
EN	input	1 bit	Input from Elastic buffer.
I_data	input	8 bits	Input Received symbols from De-Scrambler.
Un_Stripped_Data	output	256 bits	Output from Byte Unstriping. Data is Concatenated and sent to packet filter.

2.6. Packet Filter



- **Description**

  - It is the block responsible for processing the incoming data stream to ensure that the Data Link Layer (DLL) on the receiving (Rx) side receives data identical to that transmitted to the Media Access Control (MAC) layer on the transmitting (Tx) side. It accomplishes this by eliminating any framing that have been added and meticulously checking for any framing errors that may have occurred, promptly reporting such Errors to the Link Training and Status State Machine (LTSSM) upon their detection.

- **Working Mechanism**

  - It is designed to process a maximum of 32 symbols during each cycle (I_RCV_Data).

- This module is responsible for directly managing interactions with the Rx buffer.
- Upon detecting an SDP token, it systematically eliminates the two additional framing symbols, storing the remaining 6 symbols in the RX buffer by activating the o_RX_Buff W_EN signal.
- Upon encountering an STP token, it isolates the sequence number from the token, subsequently concatenating it with the TLP data. The resulting combination is then forwarded to the Rx buffer.
- If the length of the TLP suggests that the packet's termination does not occur within the current cycle, the received data is internally retained. If the packet concludes in the subsequent cycle, the data received over the two cycles are transmitted to the Rx buffer utilizing the two data buses, along with their respective indicators. However, should the packet persist beyond the current cycle, a composite of the previously retained data and portions of the newly received data is dispatched to the Rx buffer via one data bus, while the remaining bytes are held in internal buffer memory. This iterative process persists until the cycle in which the packet terminates, as this marks the point where utilization of both data buses becomes necessary.
- Upon detection of an IDL token, it is automatically disregarded and not forwarded to the DLL.
- Framing errors are identified as follows, and the o_Error signal is activated accordingly:
- Reception of an OS block (as denoted by the I_Block_Type signal) without prior reception of an EDS in the preceding data block.
- Receipt of a data block subsequent to receipt of an EDS.
- Receipt of an OS while a packet remains incomplete.
- Receipt of consecutive ordered sets within the data stream.
- (i_Buffer_R_EN) signal.

- **Ports**

Name	Direction	Size	Description
CLK	Input	1 bit	Input clock
RST_L	Input	1 bit	Asynchronous active low reset
i_RCV_Data	Input	256 bits	Output of the Byte_Unstripping
i_Symbol_Num	Input	4 bits	Indicates the number of the current received sybol in the

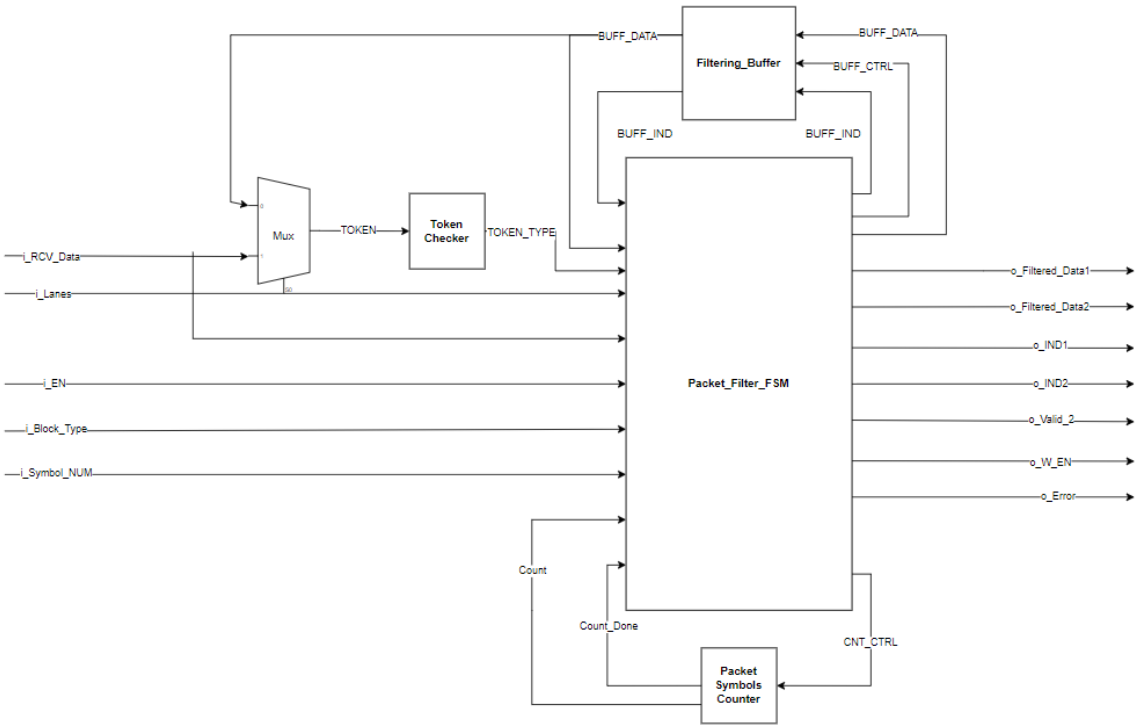
			ongoing data or OS block
i_Block_Type	Input	1 bit	Indicates whether the current block is a data block or an OS block
i_Lanes	Input	1 bit	Indicates lane configuration after link training (32 or single lane)
i_EN	Input	1 bit	Enables data processing within the packet filter when asseted
Soft_Rst_Blocks	Input	1 bit	-Input from LTSSM Controller -When asserted, it brings the Filtering fsm back to its initial state (Reset).
o_Filtered_Data1	Output	256 bits	Input de-framed data to the Rx buffer
o_Filtered_Data2	Output	256 bits	Input de-framed data to the Rx buffer
o_Last_Byte1	Output	5 bits	Packet indicator to DLL to be stored in the Rx buffer
o_Last_Byte2	Output	5 bits	Packet indicator to DLL to be stored in the Rx buffer
o_Length1	Output	11 bits	Packet indicator to DLL to be stored in the Rx buffer
o_Length2	Output	11 bits	Packet indicator to DLL to be

			stored in the Rx buffer
o_End_Valid1	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_End_Valid2	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_SOP1	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_SOP2	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_Type1	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_Type2	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_Error	Output	1 bit	Indicates the occurrence of a framing error to the LTSSM when asserted
o_Rx_Buff_W_EN	Output	1 bit	Input Write enable to the Rx buffer
o_Rx_Buff_Valid_2	Output	1 bit	Validates the seocnd data bus and its related indicators to the Rx buffer when asserted

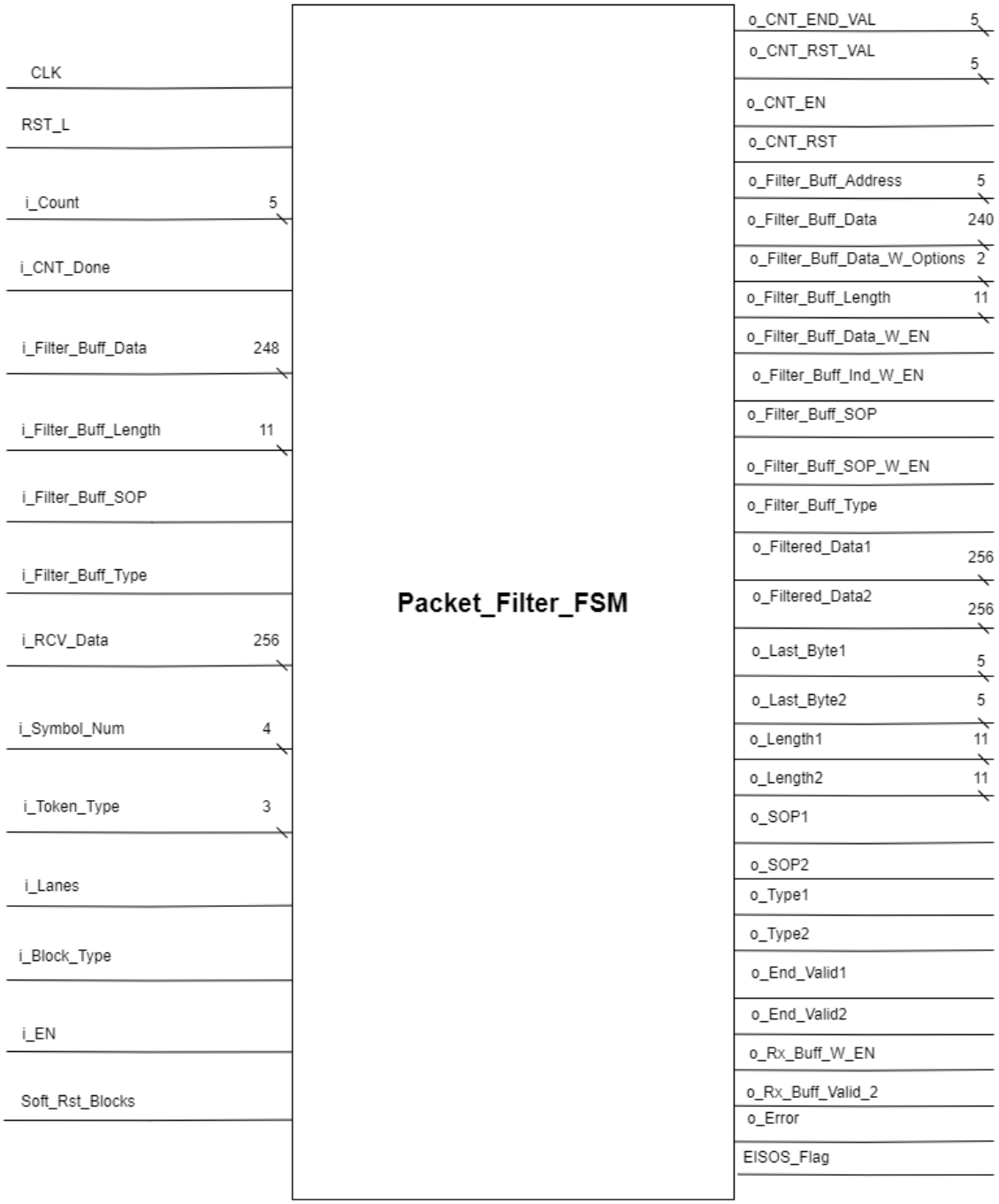
EIEOS_Flag	Output	1 bit	-Output to LTSSM Controller. -Asserted when an EIEOS is received.

- **Implementation**

  - **Packet_Filter Top**



○ Packet_Filter FSM



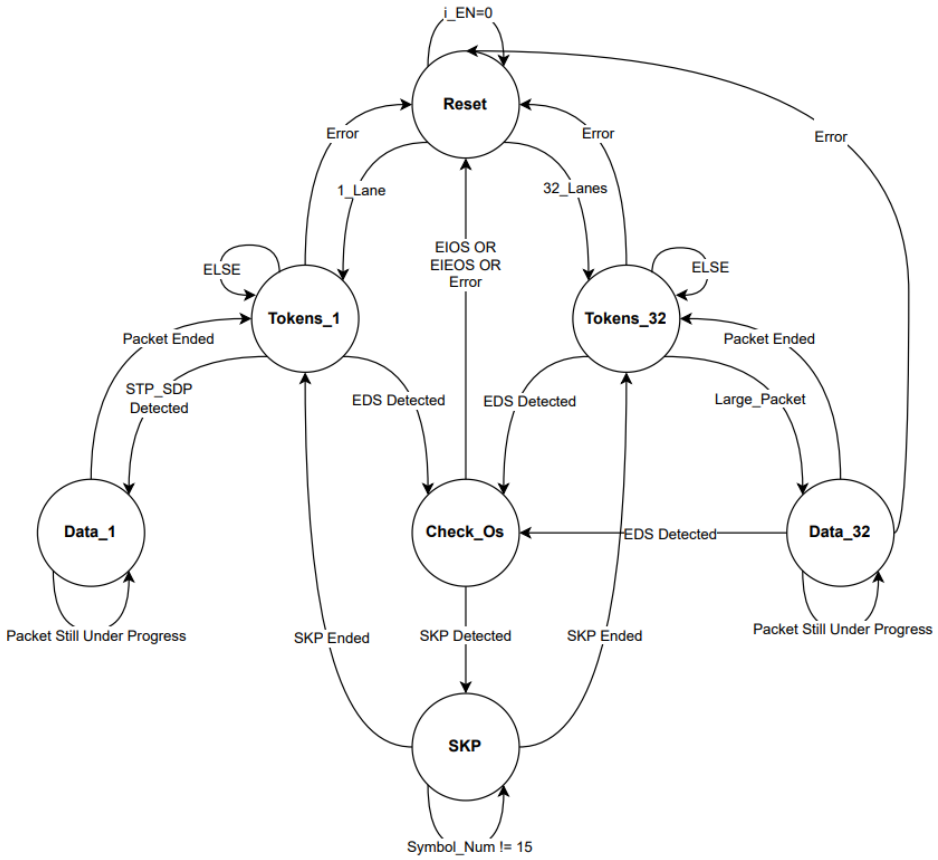
- **<u>Description</u>**

  - The Packet Filter Finite State Machine (FSM) serves as the overarching controller within the packet filter architecture. It is entrusted with the pivotal task of overseeing and managing the

diverse responsibilities inherent to the packet filter, thereby ensuring the seamless operation of the system.

- **Working Mechanism**

  - It governs the counter utilized to meticulously monitor the volume of bytes received during a one-lane configuration. This control is executed by configuring the signals (o_CNT_END_VAL, o_CNT_RST_VAL, o_CNT_EN, o_CNT_RST). Moreover, it relies on the signal I_CNT_Done to ascertain the termination of a packet, while I_Count is employed for byte addressing within the filtering buffer for writing operations.
  - It exercises precise control over the filtering_buffer, meticulously assigning values to o_Filter_Buff_Address, o_Filter_Buff_Data, o_Filter_Buff_Data_W_Options, o_Filter_Buff_Length, o_Filter_Buff_Data_W_EN, o_Filter_Buff_Ind_W_EN, o_Filter_Buff_SOP, o_Filter_Buff_SOP_W_EN, and o_Filter_Buff_Type. This assignment is contingent upon the receipt of large TLPs in a 32-lane configuration or upon the reception of DLLPs or TLPs in a one-lane configuration.
  - It adeptly employs the signal I_Token type to discern the nature of the received data, facilitating state transitions or executing pertinent actions in accordance with the identified token type.



Packet Filter Finite State Machine (FSM) States comprise seven distinct states, with certain states designated for a 32-lane configuration (Tokens_32 and Data_32), while others are allocated for single-lane operation (Tokens_1 and Data_1). The remaining states are shared between both configurations (Reset, Check_Os, SKP).

I. Reset State
   - This state represents the default state following a system reset.
   - Next state transitions are as follows:

1. Tokens_32: This transition occurs when the packet filter is enabled (as indicated by the assertion of i_EN) and operates in a 32-lane configuration (as indicated by i_Lanes).
2. Tokens_1: This transition takes place when the packet filter is enabled (as indicated by the assertion of i_EN) and operates in a single-lane configuration (as indicated by i_Lanes).
3. Reset: This transition occurs when the packet filter is disabled (as indicated by i_EN).

II. Tokens_32 State (32 Lane configuration only)

- This state represents the expectation of receiving a new packet; otherwise, it signifies the reception of a 32-byte IDLE token. It is also the state at which i_Token_Type is examined to ascertain the type of the received data.
- When receiving a Data Link Layer Packet (DLLP) or a Transaction Layer Packet (TLP) that terminates within the same cycle, the deframed packet is sent to the Rx buffer via the o_Filtered_Data1 bus.
- In the case of receiving a larger TLP, the deframed data is internally buffered.
- Any received IDLE symbols (IDLEs) are discarded.
- Next state transitions are as follows:
    1. Data_32: This transition occurs when the token type is STP, and the packet length exceeds 8 DWs, indicating that subsequent cycles will contain data related to the same TLP, making checking Token Type meaningless.
    2. Check_Os: This transition is triggered when an EDS token is observed in the last four lanes, and it is already the end of the current data block (as indicated by i_Symbol_Num).
    3. Reset: This transition occurs in the event of an unexpected error, such as the detection of an invalid or unecpected token (as indicated by i_Token_Type), or when the block type transitions to OS (as indicated by I_Block_Type). Such a transition in the Tokens_32 state is impermissible because it implies that an EDS token has not yet been observed.
    4. Tokens_32: This transition occurs when the reception of a packet ending within the same cycle transpires (a DLLP or a TLP that is 8 DWs or less in length), or when receiving IDLEs. Since all these scenarios indicate the expectation of a new packet in the next cycle, the importance of i_Token_Type is still emphasized.

III. Data_32 State (32 Lane configuration only)
- This state represents the phase where the incoming received data is to be treated as the remaining bytes of a Transaction Layer Packet (TLP) that commenced in a preceding cycle.
- Unless it is the cycle at which the TLP concludes, a combination of internally buffered data and some of the received data are to be transmitted to the RX buffer via the o_Filtered_Data1 bus. The remaining bytes of the received data are to be internally buffered.
- During the final cycle of the TLP, the same rules apply for the o_Filtered_Data1 bus, but the remaining bytes of the currently

received data are to be sent to the RX buffer over the o_Filtered_Data2 bus.

- Next state transitions are as follows:
  1. Tokens_32: This transition occurs when the packet ends, and no EDS token is observed.
  2. Check_Os: This transition takes place when the packet ends, and an EDS token is observed in the last four lanes, and it is already the end of the current data block (as indicated by i_Symbol_Num).
  3. Data_32: This transition occurs when the packet has not yet ended.
  4. Reset: Otherwise (indicating an error has occurred).

IV. Check_Os State

- This state is entered when an Ordered Set (OS) block is expected to be received.
- While in this state, if an EIEOS ID is seen, the EIEOS_Flag is asserted to be used by the LTSSM controller to make the necessary transistion to recovery.
- Next state transitions are as follows:
  1. SKP: This transition occurs when I_Block Type indicates that the currently received data is a portion of an OS, and the ordered set is an SKP ordered set.
  2. Reset: This transition occurs otherwise, which could be due to an error or the reception of EIOS or EIEOS, indicating the end of data transmission.

V. SKP State

- This state is where the reception of the remaining symbols of the SKP Ordered Set (OS) occurs, enabling data processing to resume.
- Next state transitions are as follows:
  1. Tokens_32: This transition occurs when it is already the end of the OS block (as indicated by I_Symbol_Num) and it is a 32-lane operation (as indicated by I_Lanes).
  2. Tokens_1: This transition takes place when it is already the end of the OS block (as indicated by I_Symbol_Num) and it is a one-lane operation (as indicated by I_Lanes).
  3. SKP: This transition occurs otherwise.

VI. Tokens_1 State (One Lane configuration only)

- This state denotes the anticipation of receiving a new packet. Therefore, buffering of the received bytes occurs to facilitate the transmission of a 4-byte token to the token checker. The i_Token

Type is examined after the reception of 4 bytes over 4 cycles, utilizing the filtering buffer.

- Next state transitions are as follows:
  1. Data_1: This transition occurs when an STP or SDP token is received (as indicated by i_Token_Type).
  2. Check_Os: This transition takes place when an EDS token is received, and it is already the end of the current data block (as indicated by i_Symbol_Num).
  3. Reset: This transition occurs when an invalid or unexpected token is received, indicating the occurrence of an error.
  4. Tokens_1: This transition occurs when the received token is IDL or when the 4 bytes needed for token decoding have not yet been collected (I_Count != 3).

VII. Data_1 State (One Lane configuration only)

- This state is where data processing occurs. During each cycle, the received byte is internally buffered until it is either the last byte of the currently processed packet or until 31 bytes have already been accumulated. In wither cases, the internally buffered data are combined with the currently received byte and sent to the Rx_Buffer over the o_Filtered_Data1 Bus.
- Next state transitions are as follows:
  1. Tokens_1: This transition occurs when the packet is ended.
  2. Data_1: Otherwise, the state remains unchanged.

- **Ports**

Name	Direction	Size	Description
CLK	Input	1 bit	Input clock
RST_L	Input	1 bit	Asynchronous active low reset
i_Count	Input	5 bits	Indicates the number of the currently received byte related to the packet under processing (during one lane configuration)
i_CNT_Done	Input	1 bit	Indicates that packet reception

			is done when asserted according to the assigned CNT_END_VAL
i_Filter_Buff_Data	Input	248 bits	Input data bus from the internal buffer
i_Filter_Buff_Length	Input	11 bits	Input packet indicator from the internal buffer
i_Filter_Buff_SOP	Input	1 bit	Input packet indicator from the internal buffer
i_Filter_Buff_Type	Input	1 bit	Input packet indicator from the internal buffer
i_RCV_Data	Input	256 bits	Input data bus from the Byte_Unstripping
i_Symbol_Num	Input	4 bits	Indicates the number of the current received sybol in the ongoing data or OS block
i_Token_Type	Input	3 bits	Output of the Token Checker (STP, SDP, EDS, IDL, invalid)
i_Lanes	Input	1 bit	Indicates lane configuration after link training (32 or single lane)
i_Block_Type	Input	1 bit	Indicates whether the current block is a data block or an OS block

i_EN	Input	1 bit	Enables moving forward in and keeoing in processing data when asserted
Soft_Rst_Blocks	Input	1 bit	-Input from LTSSM Controller -When asserted, it brings the Filtering fsm back to its initial state (Reset).
o_CNT_END_VAL	Output	5 bits	Input to the counter used during a one lane configuration to determine the value at which CNT_Done should be asserted
o_CNT_RST_VAL	Output	5 bits	Input to the counter used during a one lane configuration to determine the value at which it should reset when CNT_RST is asserted
o_CNT_EN	Output	1 bit	Input to the counter used during a one lane configuration to enable counting up by one.
o_CNT_RST	Output	1 bit	Input to the counter used during a one lane configuration to enable resetting

o_Filter_Buff_Address	Output	5 bits	Output byte address starting from which data should be written in the filtering buffer
o_Filter_Buff_Data	Output	240 bits	Output data bus to the filtering buffer
o_Filter_Buff_Data_W_Options	Output	2 bits	Output to the filtering buffer to determines how many bytes are to be written (1,2 or 30)
o_Filter_Buff_Length	Output	11 bits	Input packet indicator to the filtering buffer
o_Filter_Buff_Data_W_EN	Output	1 bit	Enables writing data into the filtering buffer when asserted
o_Filter_Buff_Ind_W_EN	Output	1 bit	Enables writing over the rest of the indicators (length and type) when asserted to the filtering buffer
o_Filter_Buff_SOP	Output	1 bit	Input packet indicator to the filtering buffer
o_Filter_Buff_SOP_W_EN	Output	1 bit	Enables writing over the SOP when asserted to the filtering buffer
o_Filter_Buff_Type	Output	1 bit	Input packet indicator to the filtering buffer
o_Filtered_Data1	Output	256 bits	First input data bus to the Rx buffer

o_Filtered_Data2	Output	256 bits	Second input data bus to the Rx buffer
o_Last_Byte1	Output	5 bits	Packet indicator to DLL to be stored in the Rx buffer
o_Last_Byte2	Output	5 bits	Packet indicator to DLL to be stored in the Rx buffer
o_Length1	Output	11 bits	Packet indicator to DLL to be stored in the Rx buffer
o_Length2	Output	11 bits	Packet indicator to DLL to be stored in the Rx buffer
o_SOP1	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_SOP2	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_Type1	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_Type2	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_End_Valid1	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_End_Valid2	Output	1 bit	Packet indicator to DLL to be

			stored in the Rx buffer
o_Rx_Buff_W_EN	Output	1 bit	Indicates the occurrence of a framing error to the LTSSM
o_Rx_Buff_Valid_2	Output	1 bit	Input Write enable to the Rx buffer
o_Error	Output	1 bit	Indicates the occurrence of a framing error to the LTSSM when asserted
EIEOS_Flag	Output	1 bit	-Output to LTSSM Controller. -Asserted when an EIEOS is received.
Name	**Direction**	**Size**	**Description**
CLK	Input	1 bit	Input clock
RST_L	Input	1 bit	Asynchronous active low reset
i_Count	Input	5 bits	Indicates the number of the currently received byte related to the packet under processing (during one lane configuration)
i_CNT_Done	Input	1 bit	Indicates that packet reception is done when asserted according to the assigned CNT_END_VAL

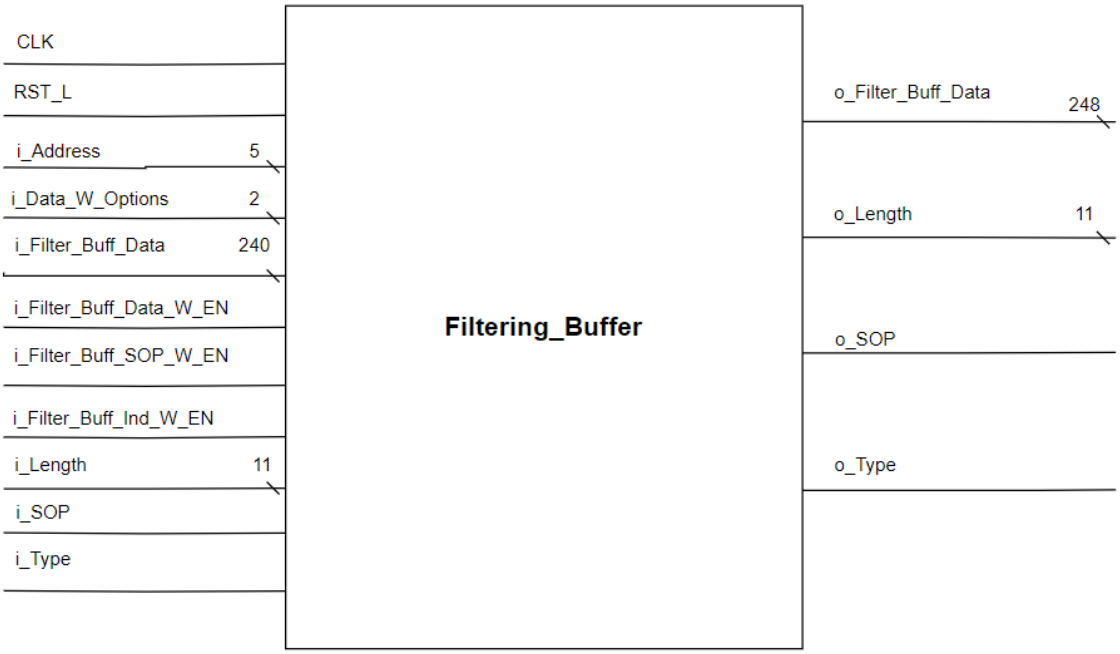
i_Filter_Buff_Data	Input	248 bits	Input data bus from the internal buffer
i_Filter_Buff_Length	Input	11 bits	Input packet indicator from the internal buffer
i_Filter_Buff_SOP	Input	1 bit	Input packet indicator from the internal buffer
i_Filter_Buff_Type	Input	1 bit	Input packet indicator from the internal buffer
i_RCV_Data	Input	256 bits	Input data bus from the Byte_Unstripping
i_Symbol_Num	Input	4 bits	Indicates the number of the current received sybol in the ongoing data or OS block
i_Token_Type	Input	3 bits	Output of the Token Checker (STP, SDP, EDS, IDL, invalid)
i_Lanes	Input	1 bit	Indicates lane configuration after link training (32 or single lane)
i_Block_Type	Input	1 bit	Indicates whether the current block is a data block or an OS block
i_EN	Input	1 bit	Enables moving forward in and keeoing in processing data when asserted

o_CNT_END_VAL	Output	5 bits	Input to the counter used during a one lane configuration to determine the value at which CNT_Done should be asserted
o_CNT_RST_VAL	Output	5 bits	Input to the counter used during a one lane configuration to determine the value at which it should reset when CNT_RST is asserted
o_CNT_EN	Output	1 bit	Input to the counter used during a one lane configuration to enable counting up by one.
o_CNT_RST	Output	1 bit	Input to the counter used during a one lane configuration to enable resetting
o_Filter_Buff_Address	Output	5 bits	Output byte address starting from which data should be written in the filtering buffer
o_Filter_Buff_Data	Output	240 bits	Output data bus to the filtering buffer
o_Filter_Buff_Data_W_Options	Output	2 bits	Output to the filtering buffer to determines how many bytes are

			to be written (1,2 or 30)
o_Filter_Buff_Length	Output	11 bits	Input packet indicator to the filtering buffer
o_Filter_Buff_Data_W_EN	Output	1 bit	Enables writing data into the filtering buffer when asserted
o_Filter_Buff_Ind_W_EN	Output	1 bit	Enables writing over the rest of the indicators (length and type) when asserted to the filtering buffer
o_Filter_Buff_SOP	Output	1 bit	Input packet indicator to the filtering buffer
o_Filter_Buff_SOP_W_EN	Output	1 bit	Enables writing over the SOP when asserted to the filtering buffer
o_Filter_Buff_Type	Output	1 bit	Input packet indicator to the filtering buffer
o_Filtered_Data1	Output	256 bits	First input data bus to the Rx buffer
o_Filtered_Data2	Output	256 bits	Second input data bus to the Rx buffer
o_Last_Byte1	Output	5 bits	Packet indicator to DLL to be stored in the Rx buffer
o_Last_Byte2	Output	5 bits	Packet indicator to DLL to be stored in the Rx buffer
o_Length1	Output	11 bits	Packet indicator to DLL to be

			stored in the Rx buffer
o_Length2	Output	11 bits	Packet indicator to DLL to be stored in the Rx buffer
o_SOP1	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_SOP2	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_Type1	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_Type2	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_End_Valid1	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_End_Valid2	Output	1 bit	Packet indicator to DLL to be stored in the Rx buffer
o_Rx_Buff_W_EN	Output	1 bit	Indicates the occurrence of a framing error to the LTSSM
o_Rx_Buff_Valid_2	Output	1 bit	Input Write enable to the Rx buffer
o_Error	Output	1 bit	Indicates the occurrence of a framing error to the LTSSM when asserted

○ Filterig_Buffer



- **Description**

  - The "filtering_buffer" serves as the internal buffer utilized within the packet filter block. It is instrumental in facilitating the storage of symbols for subsequent utilization or processing, regardless of whether the operation is conducted in a 32 or one lane configuration.

- **Working Mechanism**

  - For a 32-lane configuration:
    1. When confronted with a TLP exceeding 8 DWs in length, the entirety of the input data bus (i_Filter_Buff_Data) is utilized. This is due to the fact that during the initial cycle post-deframing, only a 30-byte data segment is deemed valid, with further data expected in subsequent cycles. Given that the Rx buffer can accommodate only 32 bytes of data at any given time, the initial 30-byte segment is stored in the filtering buffer alongside packet indicators. Subsequent cycles witness the incorporation of these 30 bytes with an additional 2 bytes, which are then stored in the Rx buffer, and the remaining 30 bytes out of the currently

recievd 32 are to be internally buffered. This iterative process continues until the packet's conclusion.

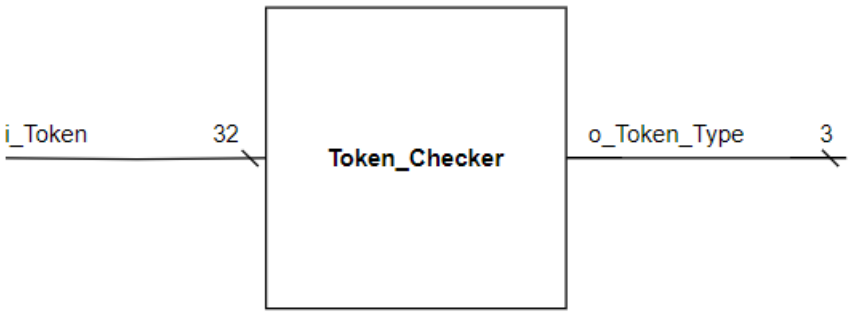
2. The default value of i_Data_W_Options is set to 2, as consistently buffering 30-byte data segments is required in this scenario.
3. The i_Filter_Buff_Ind_W_EN signal is exclusively raised during the initial cycle of TLP reception, as subsequent data segments bear identical indicators (type, length).

- For a single-lane configuration:
  1. Initially, the buffer reserves capacity to accommodate a total of 3 bytes (one byte per cycle). These three bytes are then concatenated with the concurrently received bytes to formulate a 4-byte token, subsequently forwarded to the token checker for type decoding (STP, SDP, IDL, EDS, or invalid token). Concurrently, i_Data_W_Options is configured to solely write one byte.
  2. In the event of an STP token, the two-byte sequence number is extracted and individually written to the buffer (i_Data_W_Options = 1). Subsequent cycles witness the sequential writing of each received byte to the buffer until either the packet's termination (length < 8) or the accumulation of a 31-byte data segment. Upon reaching this threshold, the o_Filter_Buff_Data is concatenated with the current received byte, and the entire 32-byte sequence is transmitted to the Rx buffer. Post-transmission to the Rx buffer, the I_Filter_Buff_SOP_W_EN signal is raised to overwrite the stored SOP with 0, as subsequent incoming data no longer pertains to the initial storage location.
  3. In the case of an SDP token, the exclusion of the two-byte DLLP frame is to happen, and data startts from byte 3. This byte, concatenated with the concurrent received byte, is individually written to the buffer (i_Data_W_Options = 1). Three out of the remaining four bytes, that are to be received, are internally buffered (one byte per cycle) and subsequently concatenated with the last received byte, then transmitted to the Rx buffer, indicating the packet's conclusion.
  4. The i_Filter_Buff_Ind_W_EN signal is raised solely during the initial cycle of TLP or DLLP reception, as subsequent data segments possess identical indicators (type, length).
  - The I_Address signal is managed by a counter tasked with tracking the buffered data post-deframing, resetting to 0 at the onset of a packet or a new token.

- **Ports**

Name	Direction	Size	Description
CLK	Input	1 bit	Input clock
RST_L	Input	1 bit	Asynchronous active low reset
i_Address	Input	5 bits	Indicates the starting byte address of the data to be writtenn
i_Data_W_Options	Input	2 bits	Determines how many bytes are to be written (1,2 or 30)
i_Filter_Buff_Data	Input	240 bits	Input data bus
i_Filter_Buff_Data_W_EN	Input	1 bit	Enables writing data into the buffer when asserted
I_Filter_Buff_SOP_W_EN	Input	1 bit	Enables writing over the SOP when asserted
i_Filter_Buff_Ind_W_EN	Input	1 bit	Enables writing over the rest of the indicators (length and type) when asserted
i_Length	Input	11 bits	input packet indicator
i_SOP	Input	1 bit	input packet indicator
i_Type	Input	1 bit	input packet indicator
o_Filter_Buff_Data	Output	248 bits	Output data bus
o_Length	Output	11 bits	Output packet indicator
o_SOP	Output	1 bit	Output packet indicator
o_Type	Output	1 bit	Output packet indicator

- **Token_Checker**



- **Description**

  - The Token Checker module is resposible for decoding the received token to determine its classification as either an STP, SDP, IDL, EDS, or an invalid token.

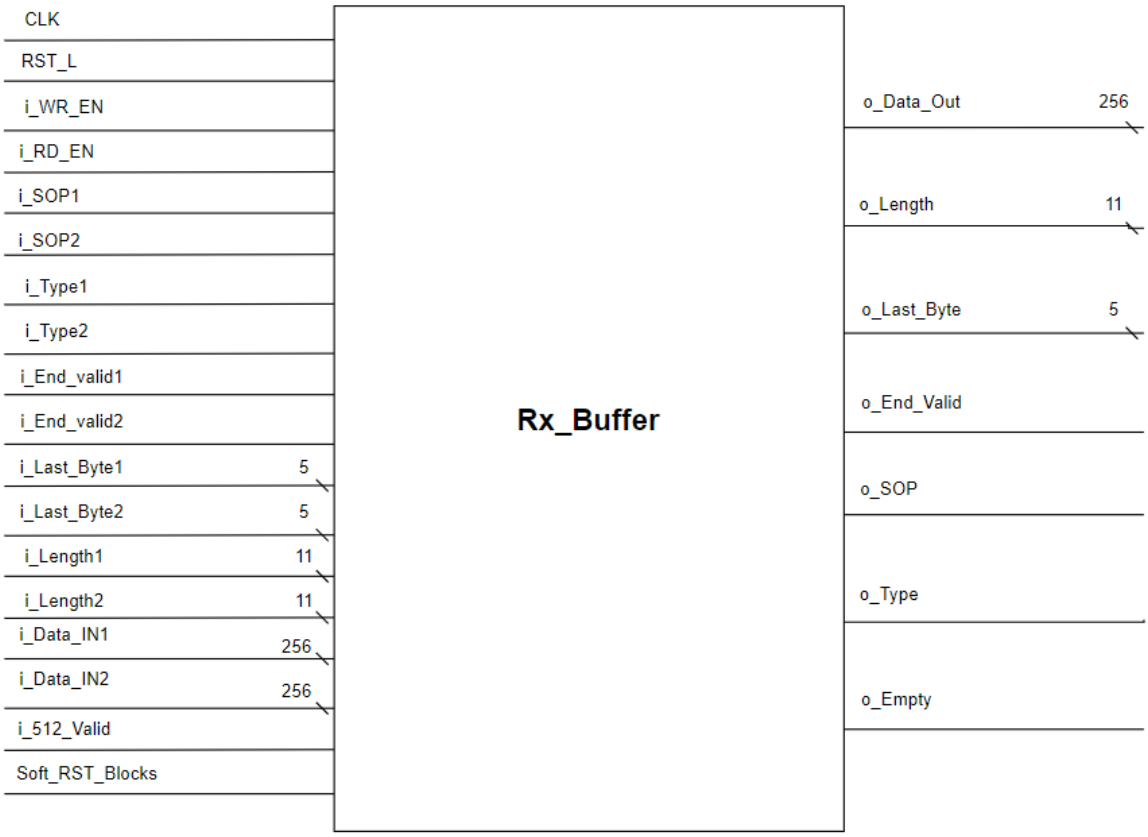
- **Working Mechanism**

  - It accepts four bytes as input (i_Token) and utilizes them to ascertain the classification of the token.
  - In the case of an SDP token, only the first two bytes of i_Token are employed for decoding, given that SDP tokens are inherently two bytes in width.
  - In a 32-lane configuration, the source of i_Token originates from the initial four bytes of the received data, which correspond to the first four lanes. This allocation is based on the design of the tx_framing module, which specifies that packets may only commence transmission from lane 0.
  - In a one-lane configuration, the source of i_Token consists of three bytes retrieved from the previously buffered data in the filtering buffer, concatenated with the presently received byte.

- **Ports**

Name	Direction	Size	Description
i_Token	Input	32 bits	Input token to be checked
o_Token_Type	Output	3 bits	Output token type to the FSM (STP,SDP,IDL,EDS,or invalid)

2.7. Rx Buffer



- **Description**

  - The RX Buffer occupies a terminal position within the MAC RX architecture, serving as the interface with the data link layer. Its primary function is to store received packets following the deframing process, thus enabling the delivery of data to the DLL in a manner

consistent with how the DLL transmits data to the MAC layer on the transmit side.

- **Working Mechanism**

  - Each storage location within the buffer is designed to accommodate 32 bytes of valid data, alongside essential control signals, known as packet indicators, intended for transmission to the DLL. These indicators include SOP (Start of Packet), Last_Byte, End_Valid, Type, and Length.
  - Data retrieval from the buffer by the DLL is conducted one location at a time, facilitated by the activation of the i_RD_EN signal.
  - The i_512_Valid signal, in conjunction with the status of the i_WR_EN signas, governs the utilization of the two data buses (i_Data_IN1 and i_Data_IN2). This determination is crucial for determining whether two consecutive locations are to be written simultaneously.
  - In a 32-lane configuration:

1. i_Data_IN1 is exclusively employed for receiving DLLPs or TLPs spanning 8 DWs or fewer in length, with i_Data_IN2 remaining unassigned.
2. Upon receipt of a TLP exceeding 8 DWs in length:
   a) During the initial cycle, neither data bus is engaged. Instead, data is internally buffered within the filtering buffer housed within the packet filter module. This is necessary as only 30 bytes of data are available after deframing, but further data reception is indicated by the TLP length.
   b) In subsequent cycles (excluding the final one), i_Data_IN1 is allocated a combination of internally buffered data and a portion of the received bytes, while the remaining received bytes are buffered internally. Consequently, i_Data_IN2 remains inactive.
   c) In the concluding cycle, i_Data_IN1 operates as in prior cycles, but the remaining received bytes are not internally buffered; instead, they are assigned to i_Data_IN2, coinciding with the assertion of i_512_Valid.
   d) The initial 30 bytes of data received in the first cycle cannot be directly stored in the RX buffer, as doing so would advance the buffer by one location (32 bytes), resulting in gaps within each location until the termination of the TLP. This deviation from the original TLP structure delivered from the DLL and inserted into the TX buffer on the transmit side. An alternative solution, apart from utilizing two data

buses, would involve modifying the RX buffer design to accommodate byte-wise advancement, thereby mitigating these gaps. However, this would introduce complexity to the RX buffer design and necessitate additional considerations in the reading operation.

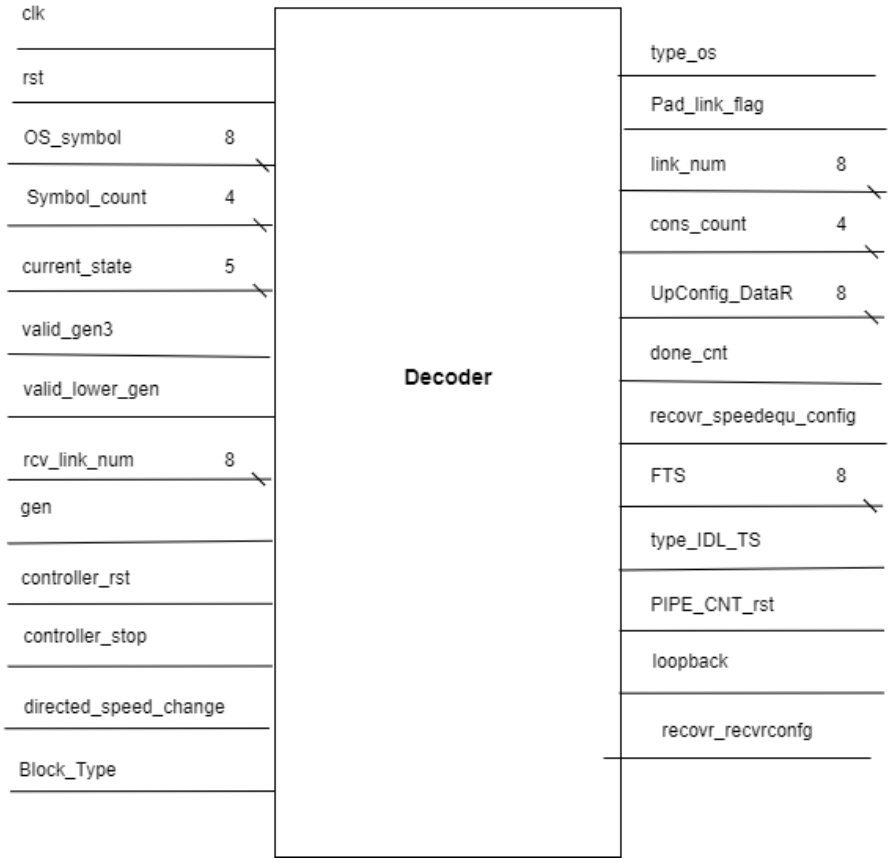
- **Ports**

Name	Direction	Size	Description
CLK	Input	1 bit	Input clock
RST_L	Input	1 bit	Asynchronous active low reset
i_WR_EN	Input	1 bit	Enables writing in the buffer when asserted
i_RD_EN	Input	1 bit	Enables reading from the buffer when asserted
i_SOP1	Input	1 bit	Input packet indicator to be stored for DLL
i_SOP2	Input	1 bit	Input packet indicator to be stored for DLL
i_Type1	Input	1 bit	Input packet indicator to be stored for DLL
i_Type2	Input	1 bit	Input packet indicator to be stored for DLL
i_End_valid1	Input	1 bit	Input packet indicator to be stored for DLL
i_End_valid2	Input	1 bit	Input packet indicator to be stored for DLL
i_Last_Byte1	Input	5 bits	Input packet indicator to be stored for DLL
i_Last_Byte2	Input	5 bits	Input packet indicator to be stored for DLL

i_Length1	Input	11 bits	Input packet indicator to be stored for DLL
i_Length2	Input	11 bits	Input packet indicator to be stored for DLL
i_Data_IN1	Input	256 bits	First input data bus
i_Data_IN2	Input	256 bits	Second input data bus
i_512_Valid	Input	1 bit	Validates the seocnd input data bus and its related indicators to the Rx buffer when asserted
Soft_Rst_Blocks	Input	1 bit	-Input from LTSSM Controller -When asserted, buffer is flushed.
o_Data_Out	Output	256 bits	Output data bus to DLL
o_Length	Output	11 bits	Output packet indicator to DLL
o_Last_Byte	Output	5 bits	Output packet indicator to DLL
o_End_Valid	Output	1 bit	Output packet indicator to DLL
o_SOP	Output	1 bit	Output packet indicator to DLL
o_Type	Output	1 bit	Output packet indicator to DLL
o_Empty	Output	1 bit	Indicates that the buffer is empty when asserted

3. LTSSM

3.1. Decoder



- **Description**

  - Decoder block is used to receive OS's Symbols from RX path and decode, analyze, check these symbols if they are matched to required OS's symbols in each state in LTSSM and also count the required number of OS's symbols in each state in LTSSM and based on that LTSSM can take the decision to move to other states.

- **Working Mechanism**

  - Decoder block is a block per lane as if we have 32 lanes so we have 32 block because each lane can carry the required OS's symbols for each state in different times due to skew difference between lanes so, we needed separate Decoder blocks.

- We check the required symbols each cycle in decoder based on **symbol_count** from RX path to know the symbol number and **current_state** from LTSSM to know the conditions of matching to enable the **cons_count**.
- After Cons_count reach to required number of counts of required symbols LTSSM stop this counter through **controller_stop.**
- After each transition in states in LTSSM, the LTSSM reset the decoder registers and counter through **controller_rst** to start from the begin in the next state.

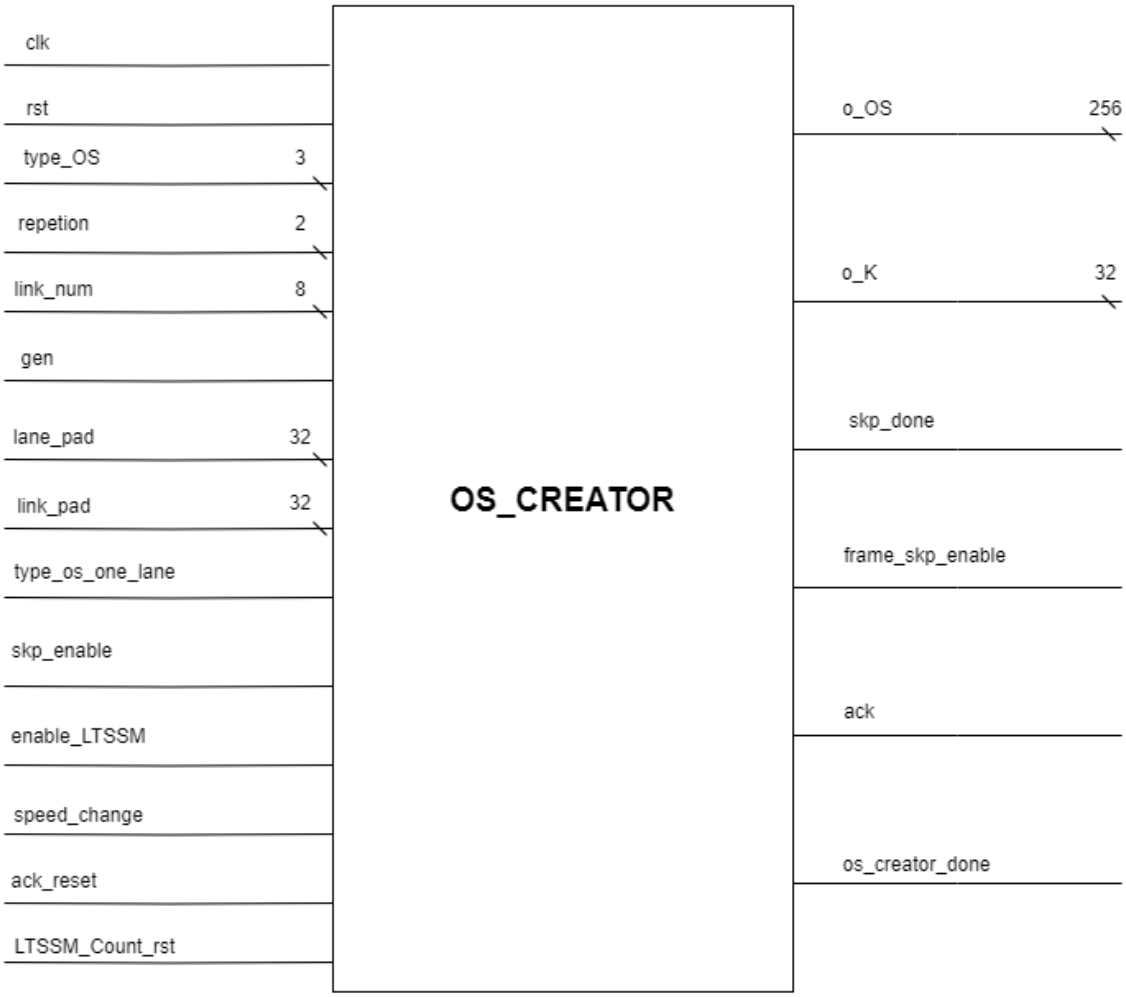
- **Ports**

Name	Direction	Size	Description
Clk	Input	1 bit	Receiver clock
Rst	Input	1 bit	Global Reset signal
OS_symbol	Input	8 bits	Input from RX path which OS symbol in each lane.
valid_gen3	Input	1 bit	**Input from PIPE** to valid incoming data in Higher Gens.
valid_lower_gen	Input	1 bit	**Input from PIPE** to valid incoming data in low Gens.
Symbol_count	Input	4 bits	**Input from RX** path to count received OS_Symbols.
current_state	Input	5 bits	**Input from LTSSM** to know the state of the FSM of LTSSM to know what I wait to receive for each state.
Gen	Input	1 bit	**Input Port from Config space** to know the current Generation.
rcv_link_num	Input	8 bits	**Input from LTSSM** contain the link number should receive as it's stored in LTSSM
controller_rst	Input	1 bit	**Input from LTSSM** to reset decoder registers and consecutive OS's counter
controller_stop	Input	1 bit	**Input from LTSSM** to stop consecutive counter of OS's as we reach to require number of counts.

directed_speed_change	input	1 bit	**Input from LTSSM** to check this variableand compare with speed change bit and enable consecutive count.
Block_Type	Input	1 bit	**Input from RX** path to indicate the type of the block Data or OS's
cons_count	Output	4 bits	**Output to LTSSM** to be able to move forward to other states as it count the required OS's to move
link_num	output	8 bits	**Output to LTSSM** to store the link number received to choose one of these link numbers and store it.
type_os	Output	1 bit	**Output to LTSSM** to indicate the type of OS as TS1 or TS2
pad_link_flag	Output	1 bit	**Output to LTSSM** to indicate none matched link number or lane number.
done_cnt	Output	1 bit	**Output to LTSSM** to indicate that we received the last required symbol in OS's for Configuration.linkwidth.start state.
recovr_speedequ_config	output	1 bit	**Output to LTSSM** to indicate that we received TS1's with directed speed change variable equal to speed change bit for Recovery.rcvlock state.
recovr_recvrconfg	Outout	1 bit	**Output to LTSSM** to indicate that speed change bit = 1 for Recovery.rcvlock state.
type_IDL_TS	output	1 bit	**Output to LTSSM** to indicate that the received symbol is IDL or TS1.
PIPE_CNT_rst	Output	1 bit	**Output to RX** path to soft reset PIPE counter that count OS_symbols.

loopback	Output	1 bit	**Output to LTSSM** to show the received loopback bit in LTSSM.
UpConfig_DataR	Output	8 bits	**Output to LTSSM** to indicate the supported data rate of the device and upconfigure bit.

3.2. OS_CREATOR



- **Description**

- The OS_CREATOR block is the main ordered sets supplier in TX architecture. It takes some control signals from the LTSSM, upon which it constructs the desired OS. It also provides feedback to the LTSSM so that it can move forward. The constructed OS is then sent to the

TX_Framing block, allowing it to be fed to other blocks without ever interrupting the data stream while operating at higher generations.

- **Working Mechanism**

- It is capable of constructing one of the following OSs (TS1, TS2, EIOS, EISOS, SDS, and Control SKP) based on the value given by type_OS. Additionally, it constructs the standard SKP when skp_enable is raised.
- It uses the repetition signal to determine how many OSs of type_OS are to be sent, allowing the ack signal to be raised accordingly to the LTSSM to move forward in the states.
- The SKP OS has a higher priority than the other OSs. When the skp_enable is raised by the SKP scheduler, it interrupts the transmission of the stream of the ordered set specified by repetition and type_OS. Once the SKP is sent (as indicated by raising skp_done to the SKP scheduler so that it deasserts the skp_enable), the stream is allowed to continue from where it was interrupted.
- The frame_skp_enable is raised during the transmission of the SKP OS to be fed to the TX_Framing so that it knows when an SKP should be sent during L0, since the OS_enable generated from the LTSSM would normally be 0 in L0.
- Even though SKP has a higher priority, it is not allowed to interrupt an OS in the middle. Therefore, the OS creator only responds to skp_enable when it is time for symbol 0 of an OS.
- The signal os_creator_done is raised during the transmission of the last symbol of each OS so that the LTSSM controller can use it to move to another state and change type_OS safely without interrupting an OS.
- When ack_reset is raised by the LTSSM controller (when making a state transition), it clears the counter responsible for running over repetition and the ack flag, allowing another stream of OS to be started.
- LTSSM_count_rst (used to reset the counter that runs over the number of transmitted symbols belonging to a specific OS) is raised by the LTSSM controller when it transitions from L0 or from being electrically idle to sending OSs back again, so that the OS transmission starts at symbol 0.
- The signals (link_num, lane_pad, link_pad, and speed_change) are only used when type_OS is TS1 or TS2.
- The signal type_os_one_lane is only used when type_OS is TS2 to validate sending TS2 with the characteristics specified by (link_num, lane_pad, link_pad, and speed_change) on all lanes (when it is de-asserted) or on only lane 0 (when asserted) while sending TS1 with pad link and lane numbers on the other lanes.

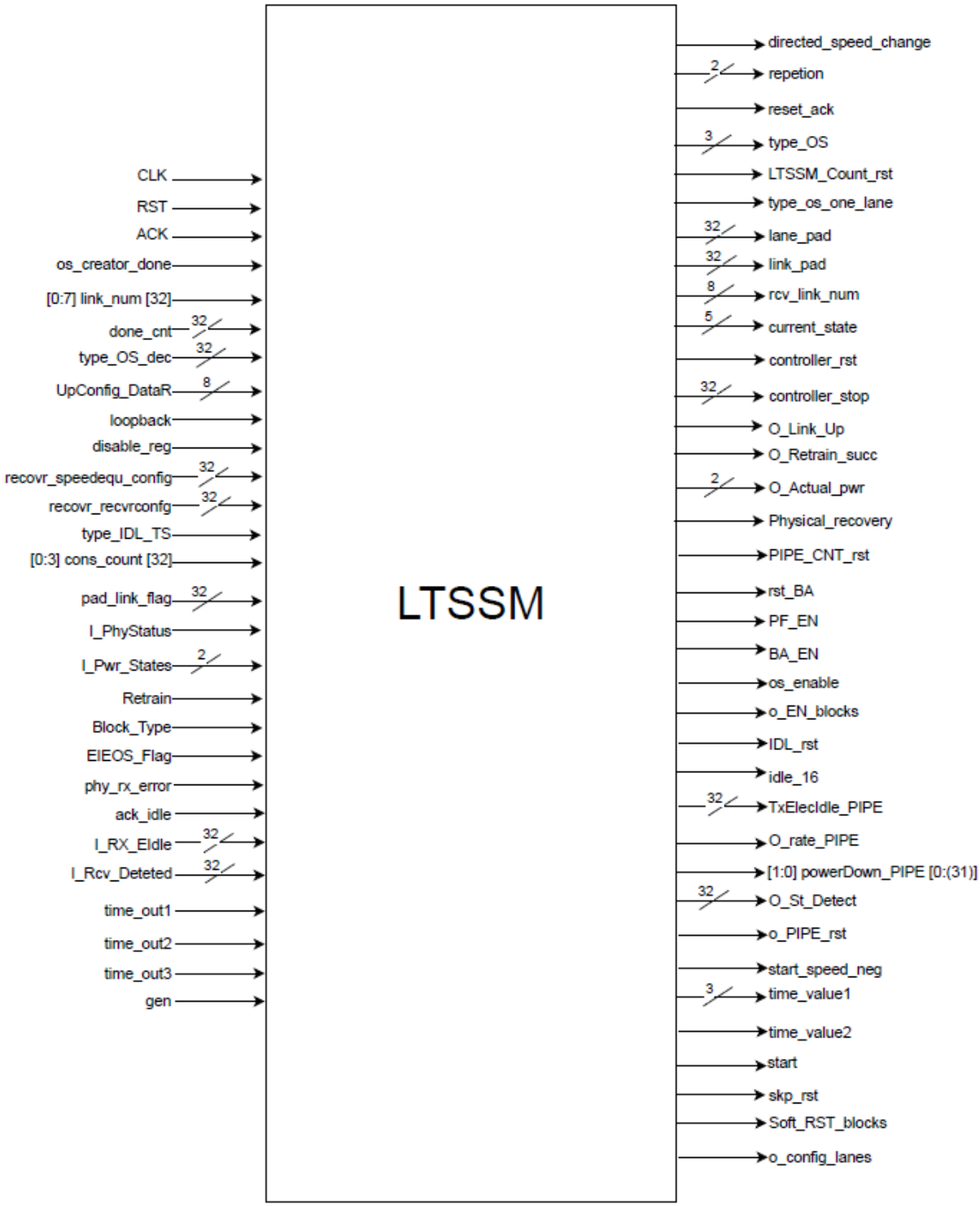
- **Ports**

Name	Direction	Size	Description
Clk	Input	1 bit	TX CLK
Rst	Input	1 bit	Global Reset signal
type_OS	input	3 bits	-Input from LTSSM -determines the OS type that is needed to be transmitted -TS1 (000) -TS2 (001) -SDS (010) -EIOS (100) -EIEOS (101) -Control SKP (011)
Repletion	input	2 bits	-Input from LTSSM -Specifies the number of times an OS is to be sent, so that the **ack** signal could be raised accordingly. - 1024, 16, 32 or unconditioned transmission
link_num	input	8 bits	-Input from LTSSM -Specifies the value of the link number filed in TS1 or TS2, to be used during link training
Gen	input	1 bit	-Specifies which generation the system is working at, so that the correct symols for the specified OS could be chosen. -Low generation (0) -High generation (1)
lane_pad	input	32 bits	-Input from LTSSM -One bit for each lane -A value of 1 corresponding to a specific lane indicates a PAD symbol is to be sent in the lane number field in TS1 or TS2. Otherwise, the lane number is to be sent.
link_pad	input	32 bits	-Input from LTSSM -One bit for each lane -A value of 1, corresponding to a specific lane, indicates a PAD symbol is to be sent in

			the link number field in TS1 or TS2. Otherwise, the link number given by **link_num** is to be sent.
type_os_one_lane	input	1 bit	-Input from LTSSM -It's only used whenn sending TS2, to specfiy whether the **type_OS** appllies for all lanes or only one lane (lane0)
skp_enable	input	1 bit	-Input from SKP_COUNTER (scheduler) -when raised, it indicates a SKP OS is to be sent.
enable_LTSSM	input	1 bit	-When asseted, it enables keeping counting on the symbols for the specified OS and the number of OSs to be sent
speed_change	input	1 bit	-Input from LTSSM -When asserted, the speed change bit in TS1 or TS2 is to be set.
ack_reset	input	1 bit	-Input from LTSSM -When asserted, it clears the counter that runs over the specified number of OSs to be sent, and the **ack** signal
LTSSM_Count_rst	input	1 bit	-Input from LTSSM -When asserted, it clears the counter that runs over each symbol in the OS
skp_done	output	1 bit	-Output to SKP_COUNTER (scheduler) -It is raised when the scheduled SKP transmisssion is finished.
frame_skp_enable	output	1bit	-Output to TX_Framing. -To be used as an indictor that a SKP OS is to interrupt the current running transmission
o_OS	output	256 bits	-Output to TX_Framing. -Represents the constructed symols for the OS (8bit symbol for each lane)
o_K	output	32 bits	-Output to Scrambler -One bit for each 8bit symbol in the **o_OS**

			-A value of 1, corresponding to a specific lane (symbol), indicates that the corresponding symbol is a control one.
Ack	output	1bit	-Output to LTSSM. -When asserted, it indicates that the tramission specified by the **repetion** is finished
os_creator_done	output	1bit	-Output to LTSSM. -When assetred, it indicates that the last symbol (normally symbol 15) of the current running OS is being transmitted

3.3. LTSSM Module



- **Description**

- The module is responsible for managing the link training and status state machine in PCIe. The module has various parameters and interfaces that allow it to interact with other components of the system, such as the Data Link Layer, Receiver (Rx), Transmitter (Tx), PIPE (Physical Interface for PCI Express), Decoder, Ordered Set (OS) Creator, Timer, and SKP (Skipping) Scheduler.

- The key functionalities of this module include:
  1. Link Management: It handles the link-up status, link retraining, and power state changes.
  2. Receiver Detection: It controls the receiver detection process and tracks the detected receiver lanes.
  3. Link Training: It manages the link training process, including sending ordered sets and configuring the link.
  4. Link Number Handling: It stores and manages the received link number during the link training process.
  5. Lane Configuration: It tracks the lanes that have received non-padded link or lane numbers and configures the lanes accordingly.
  6. State Transition: It handles the state transitions of the link training and status state machine based on various input signals and conditions.
  7. Timing Control: It utilizes various timers to manage the timing of different events during the link training and status processes.

- **Working Mechanism**

- The working mechanism of the LTSSM block is as follows:

  1. Initialization and Link Management:

  - The module initializes based on the given parameters (symbol width, state width, number of lanes).
  - It manages the link status by asserting the O_Link_Up signal when the link is ready, and the O_Retrain_succ signal when a link retraining process is successful.
  - It handles power state changes by updating the O_Actual_pwr signal based on the received I_Pwr_States.
  - It coordinates link recovery processes with the Data Link Layer by asserting the Physical_recovery signal.

2. **Receiver Detection and Lane Tracking:**

    - The module instructs the PIPE interface to start the receiver detection process using the O_St_Detect signal.
    - It tracks the detected receiver lanes using the I_Rcv_Deteted input signal from the PIPE interface.
    - It monitors the electrical idle state of the receiver lanes using the I_RX_EIdle input signal and controls the TxElecIdle_PIPE signal to force the PIPE interface into electrical idle when necessary.

3. **Link Training Process:**

    - The module coordinates the link training process by controlling the Transmitter and Ordered Set Creator modules.
    - It manages the transmission of ordered sets, such as TS1, TS2, and EIEOS, using the type_OS, repetion, LTSSM_Count_rst, and lane_pad signals.
    - It communicates with the PIPE interface to change the link rate using the O_rate_PIPE signal.

4. **Link Number Handling:**

    - The module receives the link number information from the Decoder module using the link_num signal.
    - It stores the received link number and provides it to other modules through the rcv_link_num output signal.

5. **Lane Configuration:**

    - The module tracks the lanes that have received non-padded link or lane numbers using the pad_link_flag and done_cnt signals from the Decoder module.
    - It configures the lanes accordingly and provides the controller_stop and controller_rst signals to control the operation of the decoder.

6. **State Transition and Timing Control:**

    - The module manages the state transitions of the link training and status state machine based on various input signals and conditions.
    - It utilizes internal timers and counters to manage the timing of different events during the link training and status processes.

- The current state of the state machine is made available through the current_state output signal.

- **Ports**

Name	Direction	Size	Description
CLK	input	1 bit	TX Clock
RST	input	1 bit	Global reset signal
ACK	input	1 bit	When asserted, it indicates that the transmission specified by the reptation is finished
os_creator_done	input	1 bit	When asserted, it indicates that the last symbol (normally symbol 15) of the current running OS is being transmitted
link_num	input	8 bits	Stores the link numbers received to choose one of these link numbers and store it.
done_cnt	input	32 bits	Illustrates that the last required symbol in OS's is received for the Configuration.linkwidth.start state.
type_OS_dec	input	32 bits	Indicates the type of OS whether it's TS1 or TS2
UpConfig_DataR	input	8 bits	Indicates the supported data rate of the device and the upconfigure bit.
loopback	input	1 bit	Demonstrates the received loopback bit in LTSSM.
disable_reg	input	1 bit	Indicates that the link should be disabled
recovr_speedequ_config	input	32 bit	Illustrates that the received TS1's with directed speed change variable equal to speed change bit for Recovery.rcvlock state.
recovr_recvrconfg	input	32 bit	Indicates that speed change bit = 1 for the Recovery.rcvlock state.

type_IDL_TS	input	1 bit	Indicates that the received symbol is IDL or TS1.
cons_count	input	4 bits	Comes from the decoder to enable next state advancing as the right number of counts is reached.
pad_link_flag	input	32 bit	Indicates the none matched link number or lane number.
I_PhyStatus	input	1 bit	Asserted when certain PHY functions has been completed, such as power management, receiver detection, and rate change.
I_Pwr_States	input	2 bits	The data link layer directs the physical layer to a certain power state. 00: L0 01: L0s 10: L1 11: L2
Retrain	input	1 bit	Asserted by the data link layer to instruct the physical layer into recovery to retrain the link.
Block_Type	input	1 bit	Indicates the type of the block 0: Data 1: Ordered set
EIEOS_Flag	input	1 bit	Output from PF, to be raised when an EIEOS ID is seen which means link partner has entered recovery
phy_rx_error	input	1 bit	Asserted when a receiver error is detected
ack_idle	input	1 bit	Asserted when 16 idles are sent successfully
I_RX_EIdle	input	32 bits	Asserted whenever the receiver is detected to be in electrical idle.
I_Rcv_Deteted	input	32 bits	Specifies that a receiver Detected on all lanes
time_out1	input	1 bit	Asserted when timer reach to the required number of cycles based on timeout_value1 to

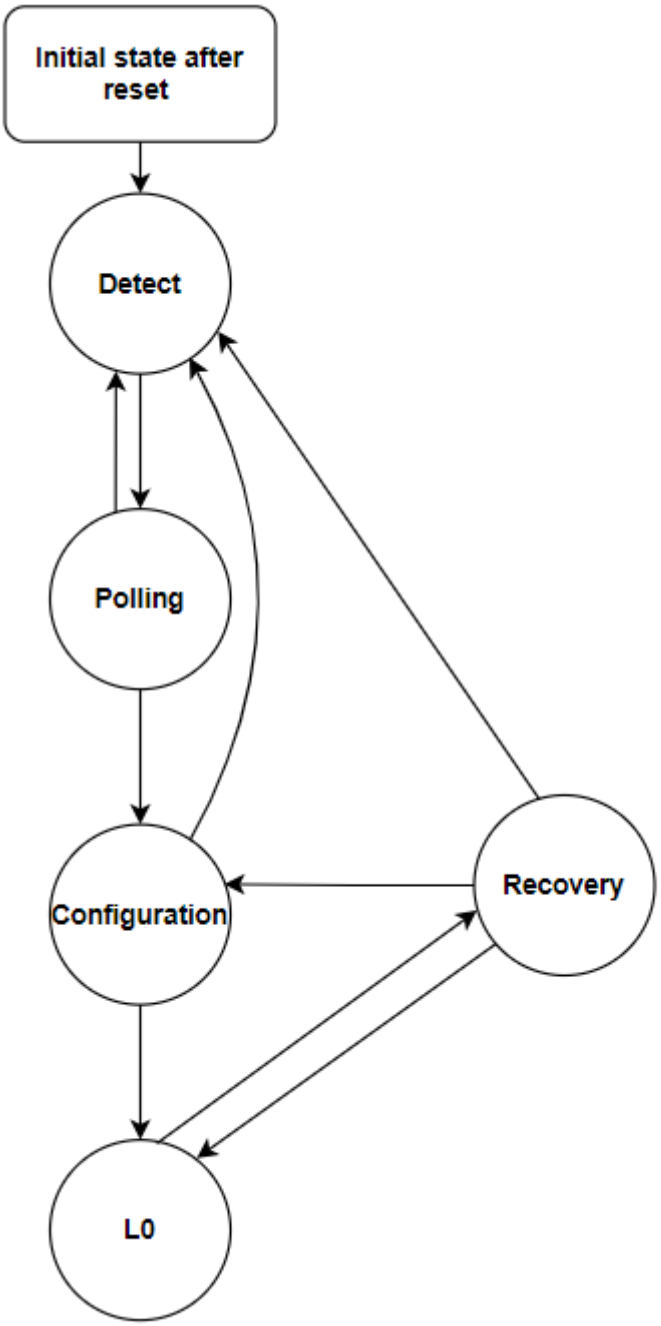
			move to another state and this used as the default time out signal for all states
time_out2	input	1 bit	Asserted when timer reach to the required number of cycles based on timeout_value2 (1ms) to move to detect quiet from detect rate change
time_out3	input	1 bit	Asserted when timer reach to the required number of cycles based on timeout_value3 (800ns) to move to Recovery_Speed_Power_Up after rate change in recovery_Speed_rate_Change
Gen	input	1 bit	Specifies which generation the system is working at, so that the correct symols for the specified OS could be chosen. -Low generation (0) -High generation (1)
directed_speed_change	Output	1 bit	When asserted, the speed change bit in TS1 or TS2 is to be set.
Repetion	Output	2 bits	Specifies the number of times an OS is to be sent, so that the ack signal could be raised accordingly. - 1024, 16, 32 or unconditioned transmission
reset_ack	Output	1 bit	When asserted, it clears the counter that runs over the specified number of OSs to be sent, and the ack signal
type_OS	Output	1 bit	Determines the OS type that is needed to be transmitted -TS1 (000) -TS2 (001) -SDS (010) -EIOS (100) -EIEOS (101) -Control SKP (011)

LTSSM_Count_rst	Output	1 bit	When asserted, it clears the counter that runs over each symbol in the OS
type_os_one_lane	Output	1 bit	It's only used when sending TS2, to specfiy whether the type_OS appllies for all lanes or only one lane (lane0)
lane_pad	Output	32 bits	One bit for each lane -A value of 1 corresponding to a specific lane indicates a PAD symbol is to be sent in the lane number field in TS1 or TS2. Otherwise, the lane number is to be sent.
link_pad	Output	32 bits	One bit for each lane -A value of 1, corresponding to a specific lane, indicates a PAD symbol is to be sent in the link number field in TS1 or TS2. Otherwise, the link number given by link_num is to be sent.
rcv_link_num	Output	8 bits	Determine the settled link number
current_state	Output	5 bits	Informs the decoder the state of LTSSM.
controller_rst	Output	1 bit	Reset the decoder registers and consecutive OS's counter
controller_stop	Output	32 bits	Stops the consecutive counter of OS's when the required number of counts is reached
O_Link_Up	Output	1 bit	Informs the data link layer that the training of the link has finished, and the link is now operational, ready to receive data.
O_Retrain_succ	Output	1 bit	Informs the data link layer the retraining has finished successfully.
O_Actual_pwr	Output	2 bits	Informs the Data Link Layer with the power state.

			00: L0 01: L0s 10: L1 11: L2
Physical_recovery	Output	1 bit	Asserted when a link recovery process is initiated by MAC layer
PIPE_CNT_rst	Output	1 bit	Resets the Pipe counter to 0.
rst_BA	Output	1 bit	Resets the Block Alignment and type detection block.
PF_EN	Output	1 bit	Enables the packet filtering block
BA_EN	Output	1 bit	Enables the Block Alignment and type detection block.
os_enable	Output	1 bit	Output to the OS_Creator to begin sending ordered sets.
o_EN_blocks	Output	1 bit	Enables Tx blocks for data and OS transmission
IDL_rst	Output	1 bit	Aims to reset the Idle count to 0.
idle_16	Output	1 bit	Asserted when 16 idles are required to be sent during link training
TxElecIdle_PIPE	Output	32 bits	Forces TX output to electrical Idle
O_rate_PIPE	Output	1 bit	Controls the link data rate.
powerDown_PIPE	Output	1 bit	Informs the pipe of the power current power state
O_St_Detect	Output	32 bits	Directs the PIPE to start receiver detection.
o_PIPE_rst	Output	1 bit	Direct the Pipe to reset state as the MAC must hold the PHY in reset until power and CLK to the PHY are stable.
start_speed_neg	Output	1 bit	Output to the timer to indicate the timer to start counting number of cycles needed for (800 ns) to move to Recovery_Speed_Power_UP
time_value1	Output	3 bits	Output to the timer to be able to determine the number of cycles to assert time_out1

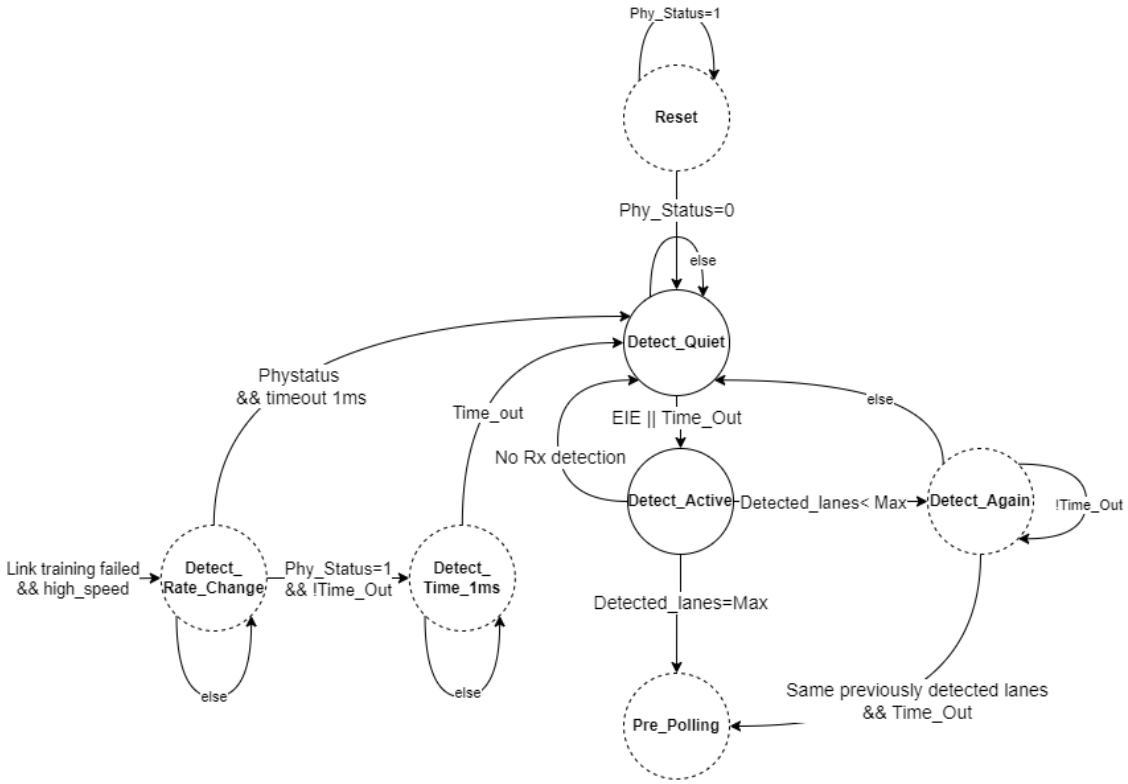
			signal in all states as this is the default input timer signal
time_value2	Output	1 bit	Output to the timer to be able to determine the number of cycles to assert time_out2 signal in detect_rate_change state.
Start	Output	1 bit	Output to the timer to start counting the time from a transition to another.
skp_rst	Output	1 bit	Aims to reset the SKP count to 0.
Soft_RST_blocks	Output	1 bit	Synchronous reset to all the blocks
o_config_lanes	Output	1 bit	Determines the lane configuration 0: 1 lane 1: 32 lanes

- **Implementation**



1. Detect

- **Detect Diagram**



- **Detect Description**

- In the Detect state, each transmitter undertakes actions to determine the presence of a receiver at the opposite end of the link.

- **Substates**

  - **Reset**

    - **Description**

      - This is the initial state after any reset, the power must be “P1”, indicating low power as the device did not start its operations yet, the transmitter must be put in Electrical idle mode as there will be no transmission until the completion of the detect state.

- **State Transition**

Exit to detect_quiet”

- if the Phystatus signal is asserted indicating that the power and the process of putting the transmitter in electrical idle is completed successfully.

➢ detect_rate_change

- **Description**

- The purpose of this state is to transition to the lowest data rate (2.5GT/s) if the transition to the detect state occurs after the data rate was increased to a speed higher than 2.5 GT/s, as the link training must be completed till L0 with the low data rate for backward compatibility, and then if a higher data rate is supported, the transition is made form L0 to the recovery state to increase the speed.
-

- **State Transition**

Exit to detect_quiet

- After the Phystatus signal is asserted indicating that the data rate has been changed to 2.5GT/s successfully, and if 1ms has already passed since entering this state.

Exit to detect_time_1ms

- After the Phystatus signal is asserted indicating that the data rate has been changed to 2.5FT/s successfully, but the time since entering this state is less than 1ms, hence the transition must be made to an intermediate state (Exit to detect_time_1ms), before proceeding to detect_quiet state, as indicated by the standard.

➢ detect_time_1ms

- **Description**

- The purpose of this state is to wait until the time of 1ms has passed since entering detect with a high data rate.

-

- **State Transition**

Exit to detect_quiet

- After the timeout signal is asserted pointing to the completion of the 1ms.

➢ detect_quiet

- **Description**

- The purpose of this state is to clear multiple variables to prepare for the next states such as: Link_Up and directed_speed_change.

- **State Transition**

Exit to detect_active

- After 12 ms timeout or when any lane exits electrical idle.

➢ detect_active

- **Description**

- The purpose of this state is to start detecting the presence of a receiver at the other end, and this is done by asserting a signal that serves to initiate the detection process in the electrical part of the layer and wait for the assertion of the Phystatus indicating the completion of the detection process successfully.

- **State Transition**

Exit to pre_polling

- If upon the completion of the detection process, all 32 lanes have detected the presence of a receiver at the other end.

Exit to detect_again

- If upon the completion of the detection process, not all 32 lanes have detected the presence of a receiver, hence there is a possibility that more lanes might detect the presence of a receiver but need more time, so the transition is made to detect_again.

➢ **detect_again**

- **Description**

  - The purpose of this state is to provide the device with an additional opportunity to detect the presence of a receiver. This process continues for 12 ms.

- **State Transition**

Exit to detect_quiet"

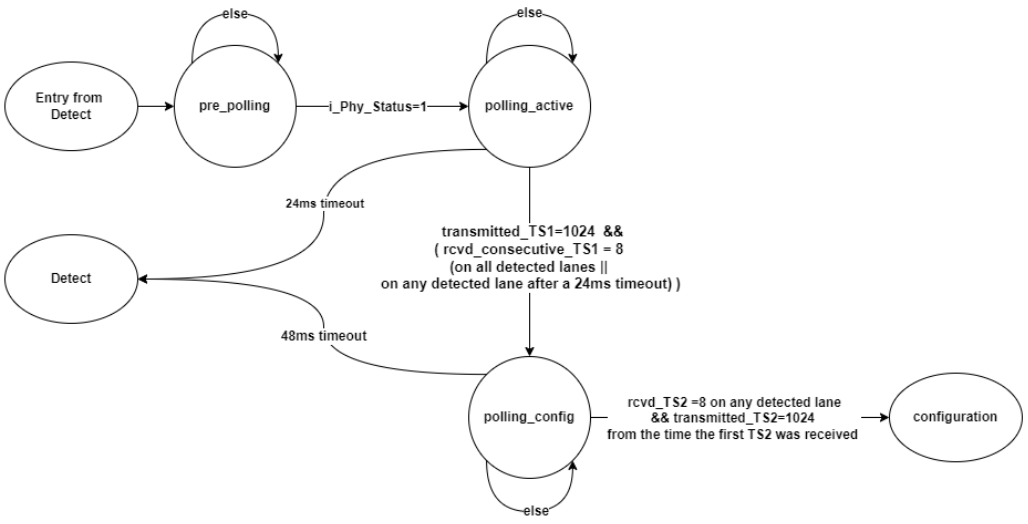
- If after timeout and the assertion of the phystatus signal, the detected lanes are different than those registered in the previous detect_active state.

Exit to pre_polling

- If after timeout and the assertion of the phystatus signal, the detected lanes are the same as those registered in the detect_active state.

2. Polling

• Polling Diagram



- **Polling Description**

  - Up to this juncture, the link has remained in the electrical idle state. However, during Polling, the LTSSM TS1s and TS2s are exchanged between the connected devices. The principal objective of this state is for the devices to mutually comprehend their communications. Specifically, they aim to achieve synchronization. Once synchronization is achieved, each device reliably receives TS1 and TS2 ordered-sets from its link partner.

- **Substate Description**

  - **Pre-Polling**

    - **Description**

      - The purpose of this substate is to bring the TX link out of the electrical idle state, enabling the transmission of TS1 ordered sets to initiate the link training process.
      - Nothing is sent during this substate as the link remains inactive. Only the power of the pipe is changed from P1 to P0.

    - **State Transition**

      Exit to Polling_Active

      - This transition occurs when the PHY status of the pipe is raised, indicating that the power has been successfully increased and the system is ready to proceed.

  - **Polling Active**

    - **Description**

      - The purpose of this substate is for each receiver to acquire bit lock and symbol lock using the received TS1 ordered sets.
      - The transmitter sends TS1 Ordered Sets with Lane and Link numbers set to PAD on all lanes that detected a receiver during the Detect phase.

- **State Transition**

Exit to Polling_Configuration:

- This transition occurs after at least 1024 TS1 Ordered Sets are transmitted, and all lanes that detected a receiver during Detect receive eight consecutive training sequences satisfying any of the following conditions:
    - TS1 with Lane and Link numbers set to PAD and the Compliance Receive bit (bit 4 of Symbol 5) is 0b.
    - TS1 with Lane and Link numbers set to PAD and the Loopback bit (bit 2 of Symbol 5) is 1b.
    - TS2 with Lane and Link numbers set to PAD.

Otherwise, after a 24 ms timout from first enetring polling_active**:**
Exit to Polling_Configuration:

- If any lane that detected a receiver during Detect receives eight consecutive training sequences satisfying any of the following conditions:
    - TS1 with Lane and Link numbers set to PAD and the Compliance Receive bit (bit 4 of Symbol 5) is 0b.
    - TS1 with Lane and Link numbers set to PAD and the Loopback bit (bit 2 of Symbol 5) is 1b.
    - TS2 with Lane and Link numbers set to PAD.
- Additionally, a minimum of 1024 TS1 Ordered Sets must be transmitted after receiving one TS1 or TS2 Ordered Set.

Else: Transition back to Detect.

➢ Polling Config

- **Description**

- In this substate, the transmitter will stop sending TS1s and begin sending TS2s. The purpose of transitioning to TS2s is to signal to the link partner that the device is ready to proceed to the next state in the state machine. This serves as a handshake mechanism to ensure that both devices on the link move through the LTSSM together. Neither device can advance to the next state until both are ready. By sending TS2 ordered sets, devices advertise their readiness. Once a device is both sending and receiving TS2s, it

confirms that it and its link partner are ready to proceed to the next state.

- The transmitter sends TS2 Ordered Sets with Link and Lane numbers set to PAD on all lanes that detected a receiver during Detect.

- **State Transition**

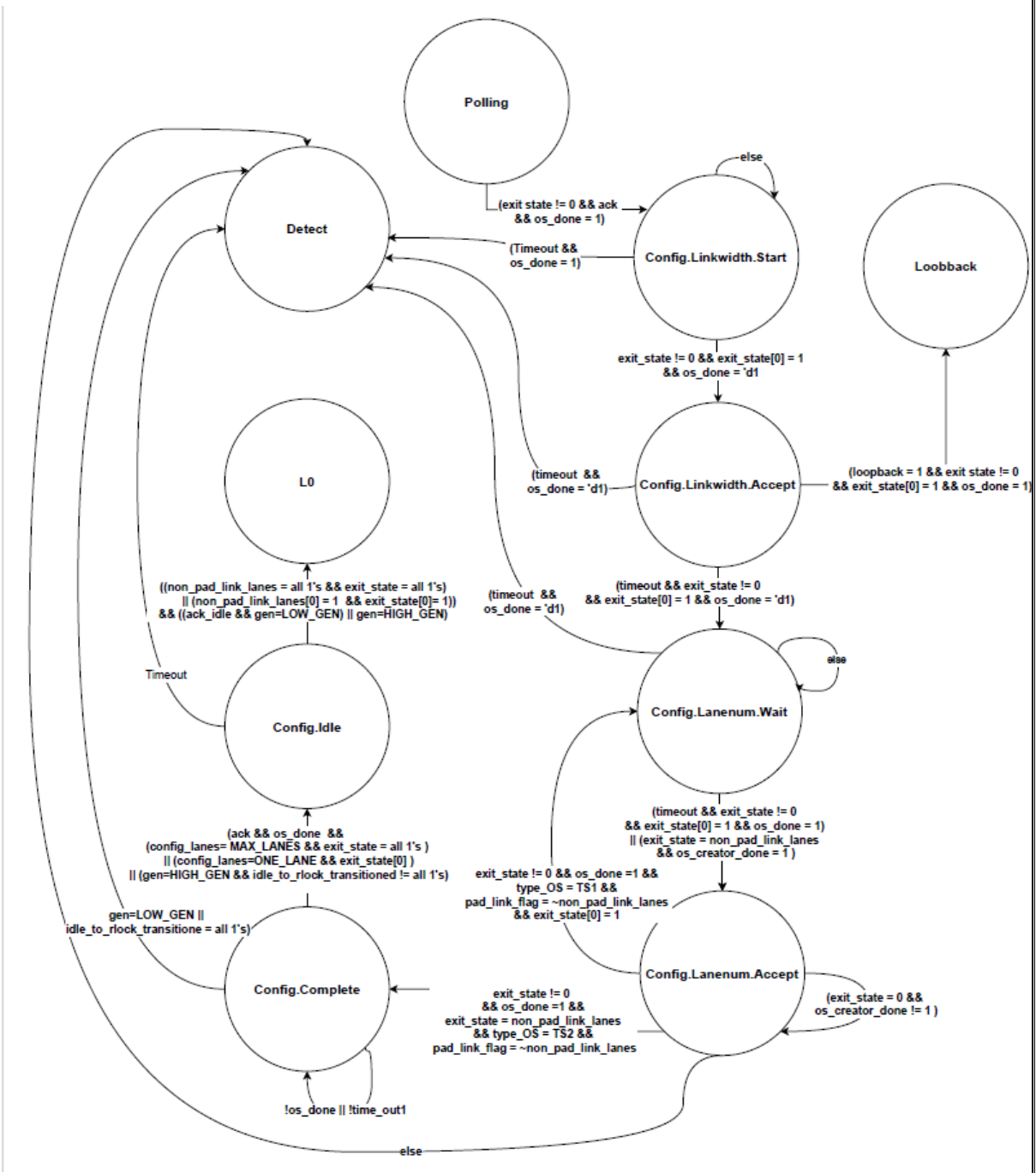
Exit to Configuration:

- This transition occurs after eight consecutive TS2 Ordered Sets, with Link and Lane numbers set to PAD, are received on any lanes that detected a receiver during Detect, and 16 TS2 Ordered Sets are transmitted after receiving one TS2 Ordered Set.

Otherwise, after a 48 ms timeout, transition back to Detect.

3. Configuration

- **Configuration Diagram**



- **Configuration Description**

  - The main goal of this state is to discover how the port has been connected and assign lane numbers for it. For example, there may be 8

lanes available, but only 2 are active, or the lanes can be split into multiple links, such as two x4 links.

- Ports have defined roles that depend on whether they are facing upstream or downstream. The description of these substates is therefore grouped into the behavior for downstream lanes and for upstream lanes.
- The downstream port (the port that transmits downstream) plays the "leader" role on this link to walk through the rest of the states in the link initialization process. The upstream port (the port that transmits upstream) plays the "follower" role.
- The leader, or downstream port, will specify the link and lane numbers to the upstream port, and the upstream port will simply reply with the same values it was told, unless there is a conflict.
- The link and lane numbers are reported in the fields of the TS1s exchanged during this time. These fields contain PAD symbols as a placeholder until actual values are assigned.

• Substate Description

➢ Config_linkwidth_start

• Description

- The purpose of this state is to settle on a link number, hence the downstream starts sending TS1s with a link number set to value between 0-255, and the upstream settles on one of the sent link numbers.
- During this state and as an upstream port, the transmitter sends TS1s with link and lane set to PAD.

• State Transition

Exit to config_linkwidth_accept

- if any detected lanes receive two consecutives TS1s with non_PAD link number and PAD lane number, the upstream must choose one of the sent link numbers and start sending TS1s with the link number set to the chosen value and PAD lane number in the config_linkwidth_accept state.

Exit to Detect

- If after 24 ms timeout, and the above condition is not satisfied.

➢ Configuration.Linkwidth.Accept

- **Description**

- After the Downstream start the configuration of link num in Configuration.Linkwidth.Start by sending TS1's with non-PAD Link number and PAD lane numbers. Upstream port accept the link configuration in this substate by sending back TS1s ordered-sets on all its Lanes that received link number with the same Link number and PAD lane numbers. After Downstream receive these TS1s with the same link number and PAD lane numbers, it starts the Lane numbers Configuration by sending TS1's wih non-PAD link numbers and actual lane numbers instead PADS.

➢ State transition

Exit to "Configuration.Lanenum.Wait"

- The next state will be Configuration.Lanenum.Wait if Upstream port received 2 consecutive TS1s with the same non-PAD link number and non-PAD lane numbers on all lanes that Transmit TS1's with non-PAD link number and PAD Lane numbers.
- Otherwise, after 2ms timeout

Exit to "Configuration.Lanenum.Wait"

- The next state will be Configuration.Lanenum.Wait if Upstream port received 2 consecutive TS1s with the same non-PAD link number and non-PAD lane number on any lane that Transmit TS1's with non-PAD link number and PAD Lane numbers.

Exit to "Detect state"

- If none of the other conditions are true, the next state will be Detect.

➢ Configuration.Lanenum.Wait

- **Description**

- After the Downstream start the configuration of lane num in Configuration.Linkwidth.Accept by sending TS1's with non-PAD Link number and non-PAD lane numbers. Upstream port accept the lane number configuration in this substate by sending back TS1s ordered-sets on all its Lanes that received non-PAD link number and non-PAD lane numbers with the same Link number and non-PAD

lane numbers that are assigned sequentially from 0 to maximum number possible for link.

- **State transition.**

Exit to "Configuration.Lanenum.Accept"

The next state will be Configuration.Lanenum.Accept if either of the 2 cases is true:

- Upstream port received 2 consecutive TS1s with same non-PAD link number and different non-PAD lane numbers from when the Lane first entered this substate on all lanes that Transmit TS1's with non-PAD link number and non-PAD Lane numbers.

- Upstream port received 2 consecutive TS2s with same non-PAD link number and non-PAD lane numbers on all lanes that Transmit TS1's with non-PAD link number and non-PAD Lane numbers.

- Otherwise, after 2ms timeout

Exit to "Configuration.Lanenum.Accept"

The next state will be Configuration.Lanenum.Accept if either of the 2 cases is true:

- Upstream port received 2 consecutive TS1s with same non-PAD link number and different non-PAD lane number from when the Lane first entered this substate on any lane that Transmit TS1's with non-PAD link number and non-PAD Lane numbers.

- Upstream port received 2 consecutive TS2s with same non-PAD link number and non-PAD lane numbers on any lane that Transmit TS1's with non-PAD link number and non-PAD Lane numbers.

Exit to "Detect state"

- If none of the other conditions are true, the next state will be Detect.

➢ Config.Lanenum. Accept

- **Description**

- In the Configuration.Lanenum.Accept state, the Upstream Port has now received either TS2 ordered sets or TS1 ordered sets from the Downstream Port. These ordered sets contain the negotiated Link and Lane numbers, rather than just the default PAD values. At this juncture, the Upstream Port must evaluate the Link and Lane

numbers that were received from the Downstream Port. The Upstream Port needs to determine whether a viable Link can be established using the Lane configuration information provided by the Downstream Port.

- **State Transition**

Exit to "Configuration.Complete":

- If the Upstream Port receives two consecutive TS2 ordered sets with the same non-PAD Link and Lane numbers
- And these numbers match the Link and Lane numbers being transmitted in the TS1 ordered sets for those Lanes
- Then the link configuration is successful, and the state machine will transition to the Configuration.Complete state.

Exit to "Configuration.Lanenum.Wait":

- If a working link can be formed using a subset of the Lanes
- The Upstream Port receives two consecutive TS1 ordered sets on those Lanes, with the same non-PAD Link number but new sequential Lane numbers
- The goal is to use a smaller group of Lanes to establish a functional link
- In this case, the state machine will transition to the Configuration.Lanenum.Wait state.
- The new Lane numbers must start from zero and increase sequentially to cover the selected Lanes
- Any Lanes that don't receive TS1 ordered sets cannot be part of the group and will disrupt the Lane numbering
- The unused Lanes must send TS1 ordered sets with PAD values for Link and Lane.

Exit to "Detect State":

- If no viable link configuration can be established
- Or if all Lanes receive two consecutive TS1 ordered sets with PAD values for Link and Lane numbers
- Then the state machine will reset and transition back to the Detect state.

➢ Config.Complete

- **Description**

- This substate within the Configuration state exclusively handles the exchange of TS2s. TS2s serve the purpose of confirming readiness between two devices on a link before proceeding to

the next state. It marks the final confirmation of the Link and Lane numbers exchanged in the preceding TS1s.

- Upon entering this substate, a device may adjust the data rates it supports and its upconfigure capability. However, it must maintain these values throughout this substate.
- If using 8b/10b encoding, Lane-to-Lane de-skew must be completed before exiting this state.
- TS2 Ordered Sets should be transmitted using Link and Lane numbers that correspond to those received in the TS1 Ordered Sets.

- **State Transition**

Exit to Configuration.Idle

- This occurs immediately after all lanes transmitting TS2 Ordered Sets receive eight consecutive TS2 Ordered Sets with matching Lane and Link numbers (non-PAD) and identical data rate identifiers (including identical Link Upconfigure Capability (Symbol 4 bit 6)). Additionally, 16 TS2 Ordered Sets must be sent after receiving one TS2 Ordered Set.

Otherwise, after a 2 ms timeout:

Exit to Detect

- If the current data rate is 2.5 GT/s or 5.0 GT/s.

Exit to Configuration.Idle

- If the idle_to_rlock_transitioned variable is less than FFh and the current data rate is 8.0 GT/s or higher.

Else

- The next state is Detect.

Notes on Transition to Configuration.Idle:

i. The changed_speed_recovery variable is reset to 0b.

ii. Lanes that are not part of the configured Link are no longer associated with the LTSSM and must transition to electrical idle.

➢ Config.Idle

• Description

- During the Configuration.Idle substate, the transmitter is sending idle data and waiting for the minimum number of received idle data so the link can transition to L0. During this time, the Physical Layer reports to the upper layers that the link is operational (Linkup = 1b).
- For 8b/10b encoding, the transmitter is sending idle data on all configured lanes. Idle data are just data zeros that get scrambled and encoded.
- For 128b/130b encoding, the transmitter sends one SDS Ordered Set on all configured lanes followed by idle data symbols. The first idle symbol on lane 0 is the first symbol of the data stream.

• State Transition

Exit to "L0 State"

- If using 8b/10b encoding, the next state is L0 if the following conditions are met:
  - 8 consecutive idle data symbol times are received on all configured lanes
  - 16 symbol times of idle data were sent after receiving one idle symbol
  - This state wasn't entered by a timeout from Configuration.Complete
- If using 128b/130b encoding, the next state is L0 if the following conditions are met:
  - 8 consecutive idle data are received on all configured lanes
  - 16 idle symbols were sent after receiving one idle symbol
  - This state wasn't entered by a timeout from Configuration.Complete

- Before data stream processing begins, lane-to-lane de-skew must be completed. The idle symbols must also be received in data blocks.
- If the software set the Retrain Link bit in the Link Control register since the last transition to L0 from Recovery or Configuration, the downstream port must set the Link Bandwidth Management bit in the Link Status register to 1b. This indicates the change was not hardware initiated (autonomous).
- The "idle_to_rlock_transitioned" variable is cleared to 00h on transition to the L0 state.

After a 2ms timeout: Exit to “Detailed Recovery Substates”

If the idle_to_rlock_transitioned variable is less than FFh, the next state is the Detailed Recovery Substates (Recovery.RcvrLock). Then:

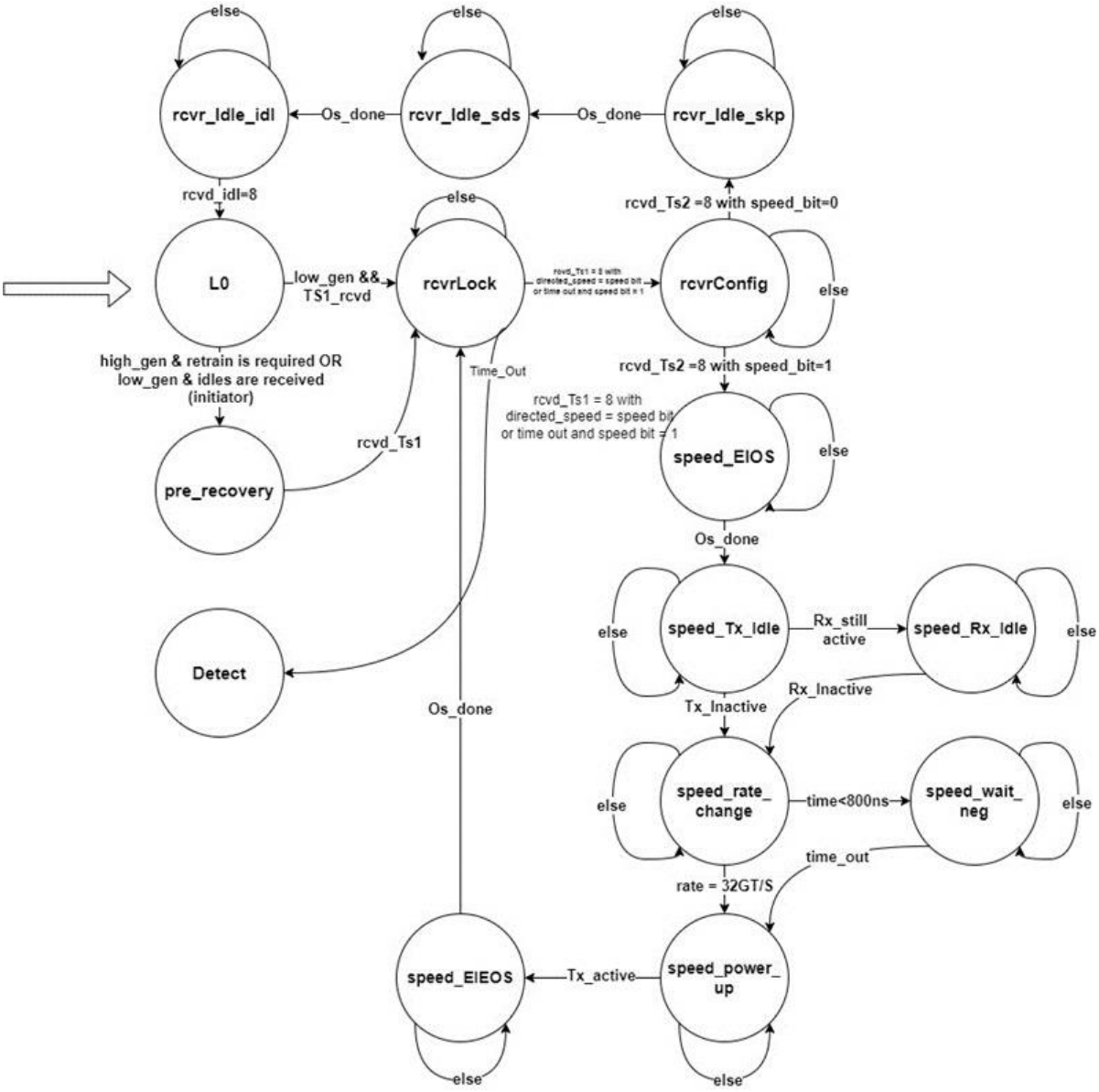
a) For an 8.0 GT/s link, increment the idle_to_rlock_transitioned variable by 1.
b) For 2.5 or 5.0 GT/s links, set the idle_to_rlock_transitioned variable to FFh.
c) The idle_to_rlock_transitioned variable keeps track of how many times the LTSSM has transitioned from the Configuration.Idle state to the Recovery state. This is because the initial sequence isn't working, likely due to improper equalization or an incompatible speed. This variable limits the number of these attempts to 256 (when the count reaches FFh) to avoid an endless loop. If the link still isn't working after 256 attempts, the next state is the Detect state, in the hopes that starting over may yield a better result.

Exit to “Detect State”

Otherwise (meaning idle_to_rlock = FFh), the next state is Detect.

4. Recovery

- **Recovery Diagram**



- **Recovery Description**

  - After Training the Link we move to L0 as normal mode and if everything works as expected so, no need to go to Recovery state but there are 2 reasons that we have to go Recovery state then return to normal mode. First once L0 is reached with a data rate of 2.5 GT/s and both devices support higher speeds, the LTSSM goes to Recovery and attempts to change the Link speed to the highest

commonly sup-ported/advertised speed, Secondly Handling Error whether Physical Layer Error in RX or Directed form Data Link Layer to retrain the link. . In this state, Bit Lock and either Symbol Lock or Block Alignment is re-acquired and the Link is de-skewed again and The Link and Lane Numbers should remain unchanged.

- Either Port can initiate Recovery when receiving IDLES in L0 by sending TS1s to neighboring device and after the 2 devices receive TS1's we enter Recovery state then proceed to other substates as shown in state diagram.

• Substate Description

➢ Pre_Recovery

• Description

- Pre.Recovery is a transit state to enter recovery whether handling speed change or handling error. If the purpose of entering the Recovery state is to change speed so we enter this substate from L0 to allow that the initiator device to send TS1s to another device before entering Recovery.RcvrLock and allow that the another device Ensure receiving TS1 then enter to Recovery.rcvrLock and If the purpose of entering the Recovery state is to Handling so we enter this substate from L0 and send EIEOS to enable Block Alignment to recover the link then entering Recovery.RcvrLock.

• State Transition

Exit to Recovery.RcvrLock

- If Low Gen (Entering to change speed), the next state will be Recovery.RcvrLock if receiving COM character symbol.
- If High Gen (Entering to handle error), the next state will be Recovery.RcvrLock if receiving ack from OS_Creator that sending 1 EIEOS.

➢ Recovery.RcvrLock

• Description

- Recovery.RcvrLock is the entrance substate of Recovery state whether Handling Error or Updating speed entrance and in this substate regardless of the speed change, Transmitters send TS1s on all configured Lanes using the same Link and Lane numbers that were set in Configuration state with speed_change bit in the Data rate identifier Symbol equal to Directed_speed_change variable. If the purpose of entering Recovery to change speed we set Directed_speed_change variable.

• State Transition

Exit to "Recovery.RcvrCfg"

- The next state will be Recovery.RcvrCfg if 8 consecutive TS1s or TS2s are received whose Link and Lane numbers match what is being sent and their speed_change bit is equal to the directed_speed_change variable.
- Otherwise, after a 24ms timeout:

Exit to "Recovery.RcvrCfg"

- The next state will be Recovery.RcvrCfg if 8 consecutive TS1s or TS2s are received whose Link and Lane numbers match what is being sent and their speed_change bit is equal to 1b.

Exit to "Detect state"

- If none of the other conditions are true, the next state will be Detect.

➢ recovery_rcvrconfig

• Description

- The purpose of this state is to verify if any remaining issues need to be addressed in the recovery state before transitioning to L0. Upon entering this state from the recovery_rcvrlock state, bit and symbol lock have already been established. If the entry into this substate was to reestablish the lock, the transition is made to

recovery_idle_skp in order to reach L0 and begin data processing. However, if there was another reason, such as changing the speed, the transition is made to recovery_speed_EIOS. Additionally, the data rates advertised by the other device are recorded in this state for potential use in future link training operations.

- During this state, the transmitter sends TS2s with link and lane number values registered from the configuration states, and with the speed bit (bit 7 in symbol 4 in the TS2) equals to the value of the directed_speed_change variable, indicating whether the current process in the recovery is intended to change the speed or not.

- **State Transition**

Exit to recovery_speed_EIOS

- if two conditions are true:

1. Any configured lanes receive eight consecutives TS2s with the lane, link numbers matching those being sent, and if the speed bit is asserted, indicating an ongoing speed change process, hence the transition is made to this state to send the EIOS to prepare the transmitter lanes to enter electrical idle, and inform the other device.
2. 32 TS2s have been sent since receiving one TS2 with specified features in condition one.

Exit to recovery_idle_skp

- if two conditions are true:

1. any configured lanes receive eight consecutives TS2s with the lane, link numbers matching those being sent, and if the speed bit is deasserted, indicating that there is no speed change process, and the transition must be made to recovery_idle_skp state to reach the L0 state and start data processing.
2. 16 TS2s have been sent since receiving one TS2 with specified features in condition one.

- The directed_speed_change and the changed_speed_change variables are cleared upon the transition to indicate that the process of the speed change is done and prepare them to take the correct values if the recovery state is entered later again for other speed changing reasons.

Exit to Config_linkwidth_start

- if any lanes received 8 consecutives TS1s with different link and lane number, than those recorded since completion of configuration state, indicating that the other device has made the transition to the configuration state, to reconfigure the link width, due to inappropriate operation with the current settings.

Exit to detect

- If after 48 ms timeout and none of the above conditions is true.

➢ Recovery_Speed_EIOS

• Description

- Recovery.Speed.EIOS is the entering substate to change the rate of the link by preparing the 2 devices for that change as we send 1 EIOS to inform another device that we will enter the IDLE state.

• State Transition

Exit to Recovery.Speed.TX.IDLE

- The next state will be Recovery.Speed.TX.IDLE if we receive ack from OS_Creator that Sending 1 EIOS.

➢ Recovery_Speed_TX_IDLE

• Description

- Recovery.Speed.TX.IDLE is the substate that we Enter TX in IDLE state by Power down the link and wait for ack from PIPE that we are in IDLE state and remain in this IDLE state until Ensuring that RX device is in IDLE state so we check this before Exit from IDLE state so, after receiving the ack from PIPE that we are in IDLE state then we check the state of RX device if IDLE state so we move to change the rate directly and if not we go to another state to wait for RX device go to IDLE state.

- **State Transition**

Exit to Recovery.Speed.Rate.Change

- The next state will be Recovery.Speed.RX.IDLE if we receive ack from PIPE that we are in IDLE state and the RX device in IDLE state through I_RX_EIdle = {‘d32{1’b1}}.

Exit to Recovery.Speed.RX.IDLE

- The next state will be Recovery.Speed.RX.IDLE if we receive ack from PIPE that we are in IDLE state and the RX device not in IDLE state through I_RX_EIdle != {‘d32{1’b1}}.

Exit to Detect state

- The next state will be Detect if the Recovery speed state Time (48 ms) is out.

➢ Recovery_Speed_RX_IDLE

- **Description**

- Recovery.Speed.RX.IDLE is the substate that we wait RX device until enter in IDLE state after TX sending EIOS and entered IDLE state and after ensuring that Rx device enter in IDLE state we move forward to Recovery_Speed_Rate_Change.

- **State Transition**

Exit to Recovery.Speed.Rate.Change

- The next state will be Recovery.Speed.RX.IDLE if the RX device enter in IDLE state through I_RX_EIdle = {‘d32{1’b1}}.

Exit to Detect state

- The next state will be Detect if the Recovery speed state Time (48 ms) is out.

➢ Recovery_Speed_Rate_Change

- **Description**

- Recovery_Speed_Rate_Change is the substate that actually we change the rate using the O_rate signal to PIPE and wait until receiving ack from PIPE that rate is already changed and also check if the time needed to power up is timed out or not as if time is out we

move directly Recovery_Speed_Power_Up and if not we move to Recovery_Speed_Wait_Neg to wait for this time.

- **State Transition**

Exit to Recovery.Speed.Wait.Neg

- The next state will be Recovery.Speed.Wait.Neg if we receive ack from PIPE that the rate is already changed through I_PHY_Status = 1. And time needed to power up (800ns) hasn't been finished yet.

Exit to Recovery.Speed.Power.Up

- The next state will be Recovery.Speed.Power.Up if we receive ack from PIPE that the rate is already changed through I_PHY_Status = 1. And time needed to power up (800ns) has been finished.

Exit to Detect state

- The next state will be Detect if the Recovery speed state Time (48 ms) is out.

➢ Recovery_Speed_Power_Up

- **Description**

- Recovery.Speed.Power.Up is the substate that we Power up the TX device to exit IDLE state and return to normal mode through PowerDown_PIPE = P0 and wait for ack from PIPE that we already powered up the TX device.

- **State Transition**

Exit to Recovery.Speed.EIEOS

- The next state will be Recovery.Speed.EIEOS if we received ack from PIPE that the power is up.

Exit to Detect state

- The next state will be Detect if the Recovery speed state Time (48 ms) is out.

➢ Recovery_Speed_Wait_Neg

- **Description**

- Recovery.Speed.Wait.Neg is the substate that we wait for the needed time which is 800ns to be able to Power up the TX device then allow that for sending EIEOS so, after time out we move forward to Recovery.Speed.Power.Up

- **State Transition**

Exit to Recovery.Speed.Power.Up

- The next state will be Recovery.Speed.Power.Up after 800 ns timeout.

Exit to Detect state

- The next state will be Detect if the Recovery speed state Time (48 ms) is out.

➢ Recovery_Speed_EIEOS

- **Description**

- Recovery.Speed.EIEOS is the substate that we send 1 EIEOS to enable Block Alignment and and allow that RX to be in normal mode and wait for the Ack from OS_Creator then move to Recovery.RcvrLock.

- **State Transition**

Exit to Recovery.RcvrLock

- The next state will be Recovery.RcvrLock if we received ack from OS_Creator that we already sent 1 EIEOS.

Exit to Detect state

- The next state will be Detect if the Recovery speed state Time (48 ms) is out.

5. L0

- **Description**

- L0 is the normal mode state as in this state we send Packets received from Data link layer if it's available and enter L0 state after Training process with data rate 2.5 GT/s so, we can upgrade the rate to common higher data rate by moving to recovery state from L0 and also, in normal mode if any RX error in physical layer happens or directed from data link layer using retrain signal we move to recovery.

- **State Transition**

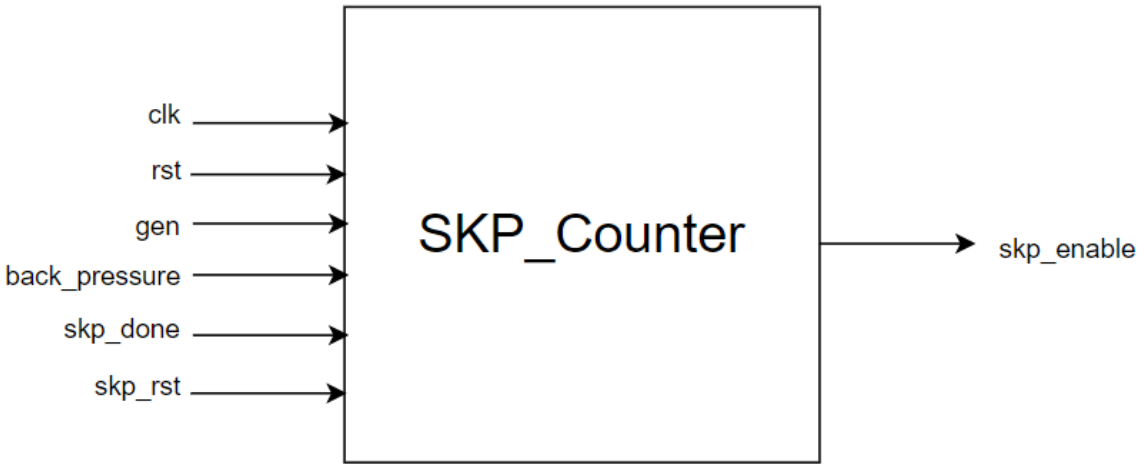
Exit to Pre.Recovery

- If LOW GEN if we received IDL not COM character (TS1 Symbol(0)) as **type_IDL_TS** = 0.
- If HIGH GEN if any error happened (retrain = 1 or physical_Recovery = 1).

Exit to Recovery.RcvrLock

- If LOW GEN if we received COM character (TS1 Symbol(0)) as **type_IDL_TS** = 1.

3.4. SKP Counter



- **Description**

- The SKP_Counter module is essential for counting the number of cycles between SKP transmission in high and low GENs. In lower generations, the transmitter schedules a SKP ordered set transmission once every 1180 to 1538 symbol times while in higher generations, the transmitter schedules SKP ordered set once every 370 to 375 blocks, thus the allowed time between SKP transmission equals 6000 Symbol times.

- **Working Mechanism**

- The working mechanism of the SKP Counter block is as follows:
  1) Asserting the skp_enable to notify the OS_Creator module for time to send the SKP ordered set according to the generation.
  2) Checking if the OS_Creator has already sent the SKP ordered set using skp_done.
  3) Reseting the counter in special states as directed by the LTSSM using skp_rst.

- **Ports**

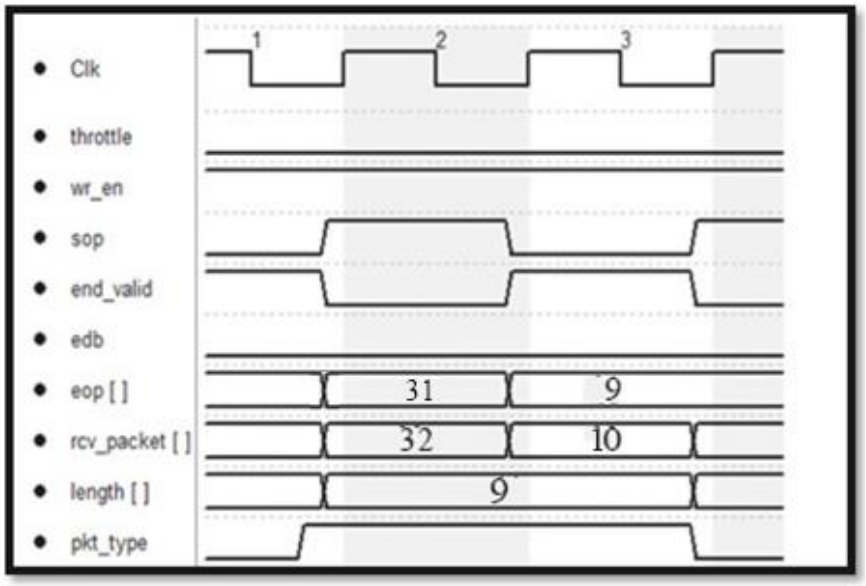
Name	Direction	Size	Description
Clk	Input	1 bit	TX Clock
rst	Input	1 bit	Global reset signal
back_pressure	Input	1 bit	The back pressure coming from the buffer logic.
gen	Input	1 bit	The current generation. 1: High Generation 0: Low Generation
skp_done	Input	1 bit	To confirm that the OS_Creator has sent the SKP already
skp_rst	Input	1 bit	Signal from the LTSSM to reset the SKP count to 0
skp_enable	Output	1 bit	To notify the OS_Creator module to send the SKP OS

Timing diagrams

1. Tx Timing Diagrams:

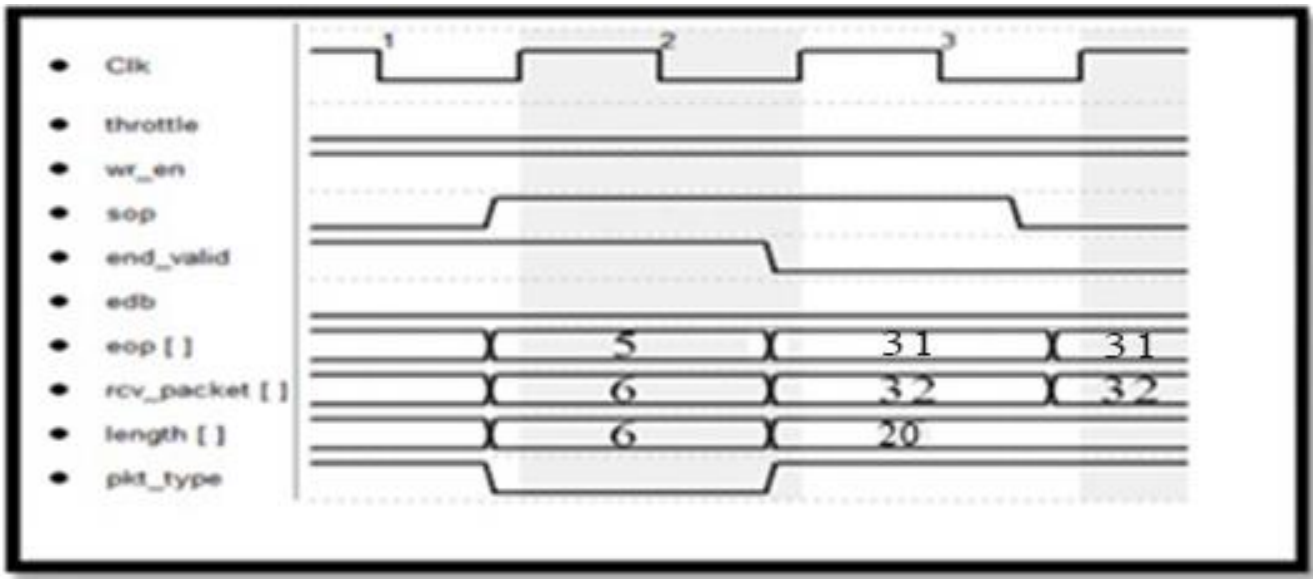
1.1. Receiving TLP Case

- **In this case we show that when receiving TLP Large packet as we receive it over 2 cycles as the length after framing is 44 Bytes as Transaction layer send the packet with length 9 DW and this packet is TLP as type = 1, last Byte = 19 and end valid = 1 in the last location.**



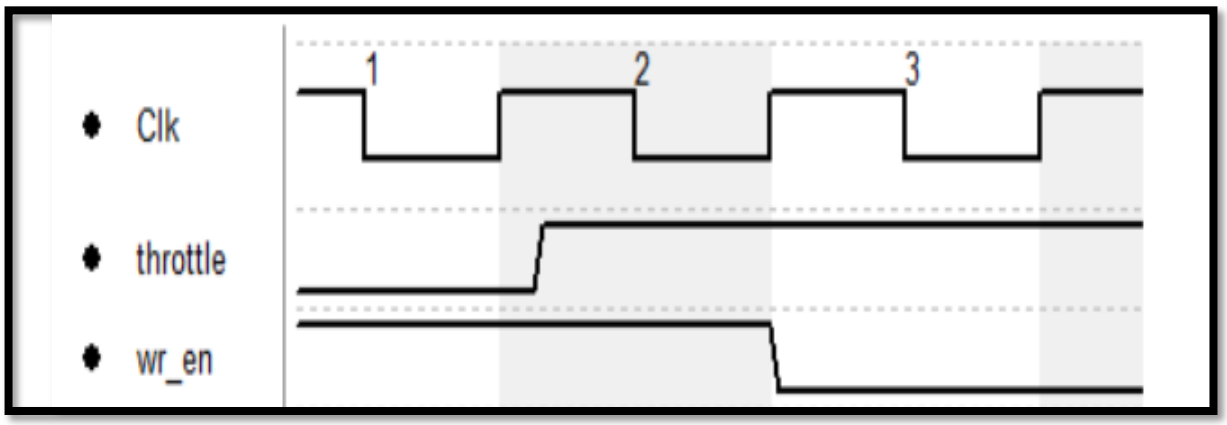
1.2. RECEIVING DLLP then TLP case

- **In this case we show that when receiving DLLP packet as we receive it over 1 cycle as the length after framing is 8 Bytes and this packet is DLLP as type = 1, last Byte = 5 and end valid = 1 in the last location then sop = 1 so, it's the beginning of another TLP packet as type = 1 and we receive it over cycles as the length = 20 DW, end valid = 1 in the last.**



1.3. Full Buffer case

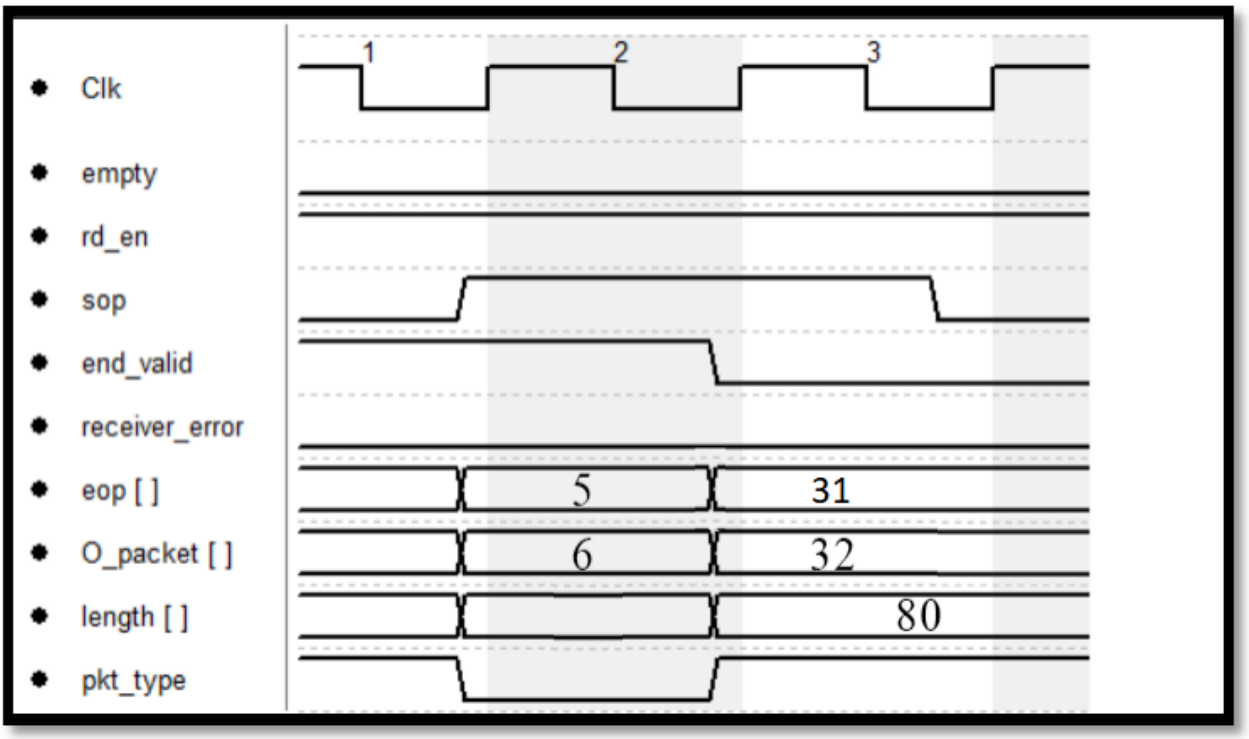
- **In this case we show throttling case when we throttle Data link layer with throttle = 1 when TX Buffer is full so wr_en = 0.**



2. Rx Timing Diagrams:

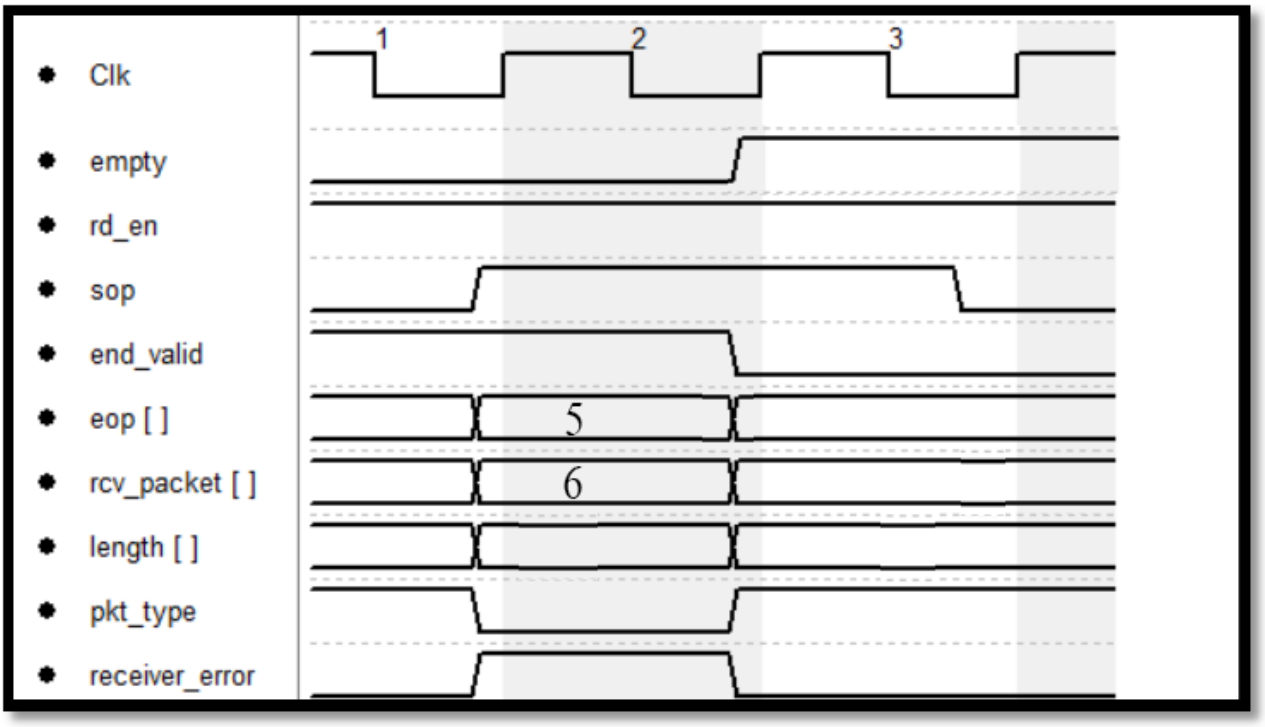
2.1. Sending DLLP then TLP to DLL

- The start of the DLLP is indicated by raising the sop signal.
- Since the interface bus is 32 bytes wide and the DLLP is always 6 bytes, the end_valid signal is also raised along with the sop in the same cycle, and the eop takes a value of 5 (0 to 5 == 6 bytes).
- The pkt_type signal is low, indicating the delivery of a DLLP.
- The Length field is not of important value in this case since the length indicator is supposed to mention the length in DW for only TLPs.
- O_packet = 6 during this transmission, which means that only 6 bytes out of the 32-byte interface bus are of important value.
- In the following cycle, a TLP (indicated by the assertion of pkt_type) of length 80 DWs is to be delivered.
- This is a large packet that should be sent to the DLL over 11 cycles.
- The start of the TLP is indicated by keeping the sop asserted, but this time the end_valid is de-asserted since the packet is not to end before another 10 cycles.
- The eop takes the value of 31, and the O_packet takes the value of 32 over the remaining cycles until the one at which the packet concludes.
- The empty signal is kept de-asserted since packets are being written into the buffer.



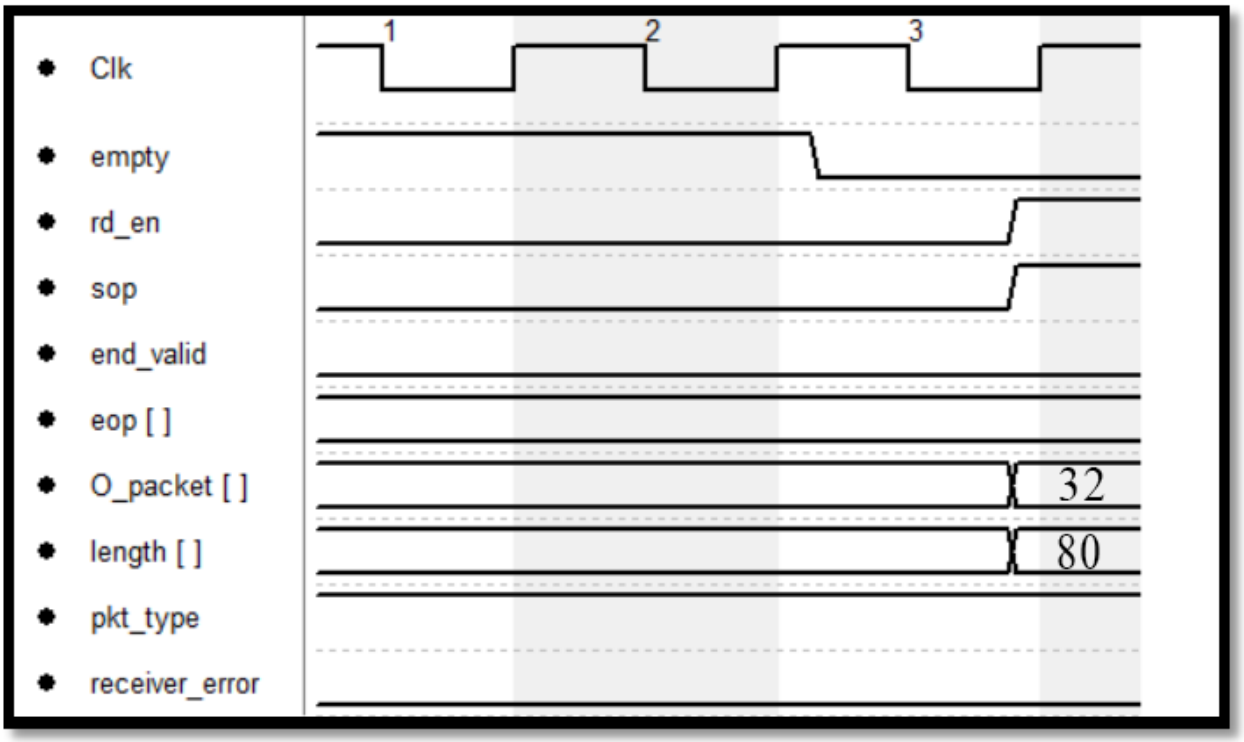
2.2. Error Reporting case

- This case represents a receiver error that is discovered during a DLLP delivery to the DLL.
- The error is indicated by asserting the receiver error signal.
  Upon the receiver error assertion, the DLL is to neglect any packet delivered, which is the DLLP in this case, and the buffer is to be flushed, indicated by the assertion of the empty signal.



2.3. Empty Buffer case

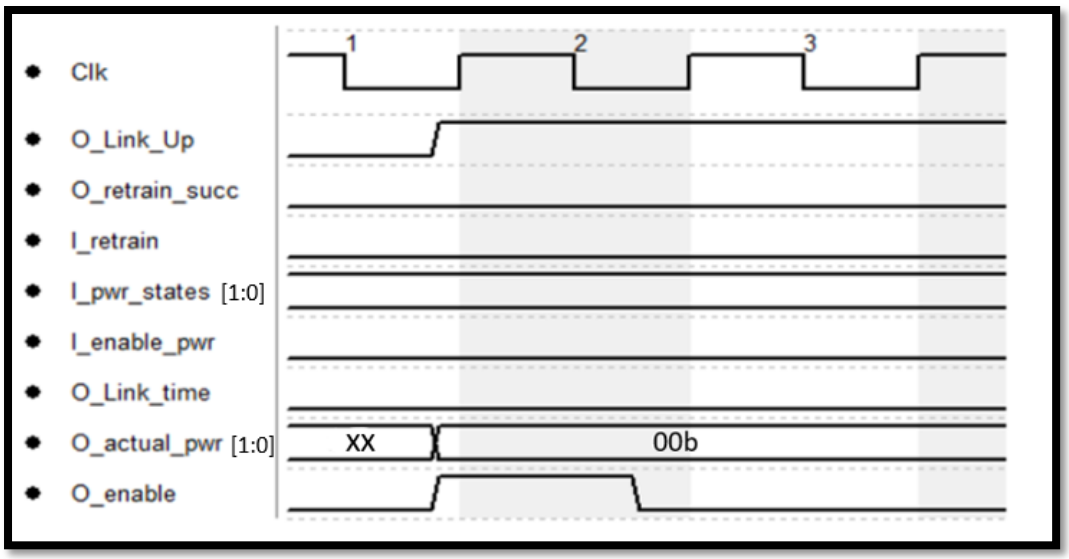
- In this case, the MAC layer wasn't receiving any packets to be delivered to the DLL, so the empty signal is asserted, and the rd_en signal from the DLL is de-asserted.
- Then, the start of a TLP of length 80 DWs is written into the buffer, which is indicated by the de-assertion of the empty signal.
- When the DLL detects the de-assertion of the empty signal, it asserts the rd_en signal.



3. LTSSM Timing diagrams:

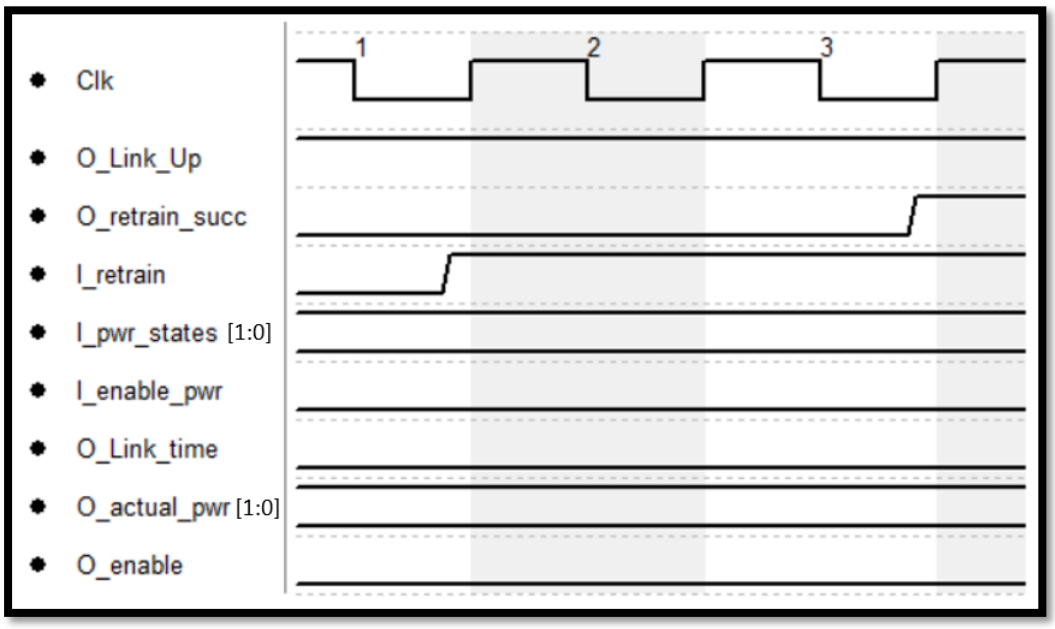
3.1 Link UP assertion after training

- After Training Process completion, the power state is p0, and the Link_Up is asserted to allow the data link layer to send packets



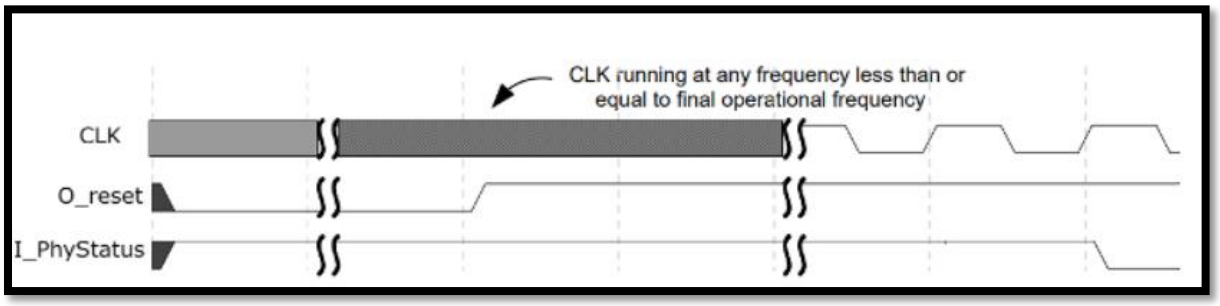
3.2 Retrain

- In case the Data link layer kept on receiving the same packet incorrect multiple times, that indicates that the link need to be retrained, and the Retrain signal is asserted



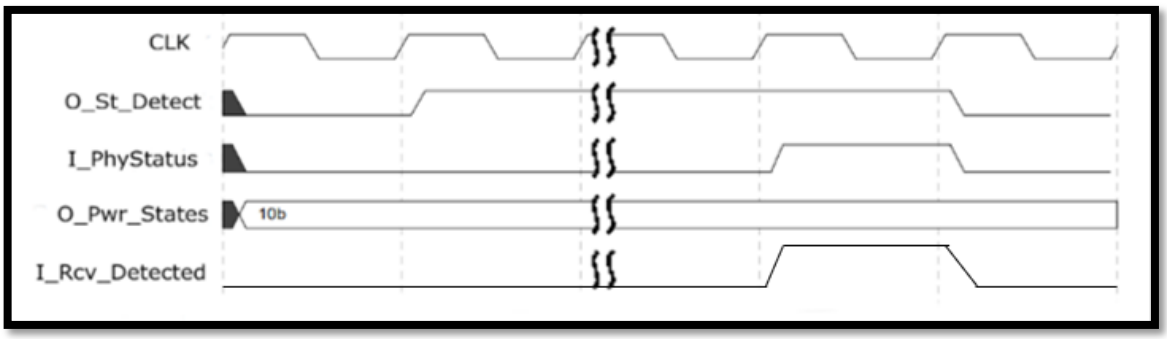
3.3 After Reset

- At the beginning the I_Phystatus signal is asserted until the clk is stable, then the I_Phystatus signal is de-asserted to allow the operation of the system



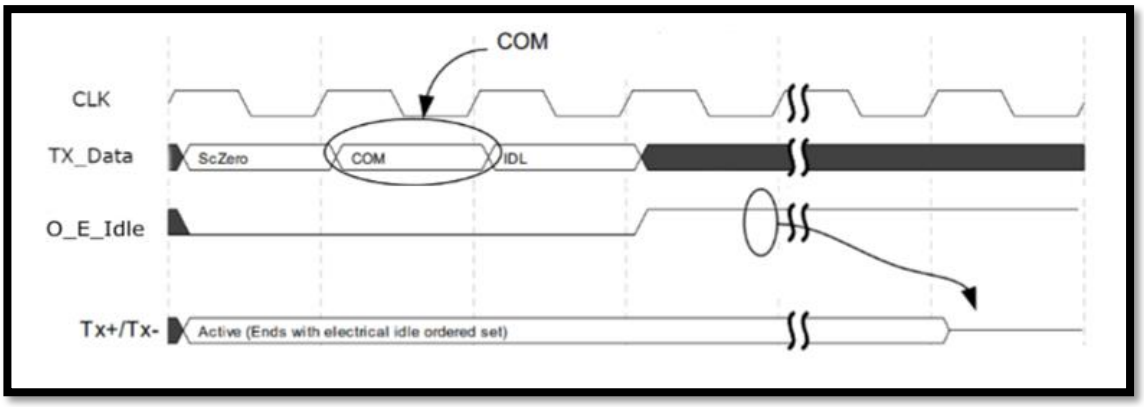
3.4 Receiver Detection

- In the Detect state, and as directed by the LTSSM, the signal o_st_detect is asserted to allow the start of the detection process, and the response when the I_Phystatus signal is asserted, the I_Rcv_Detected is checked to see whether a receiver is detected or not.



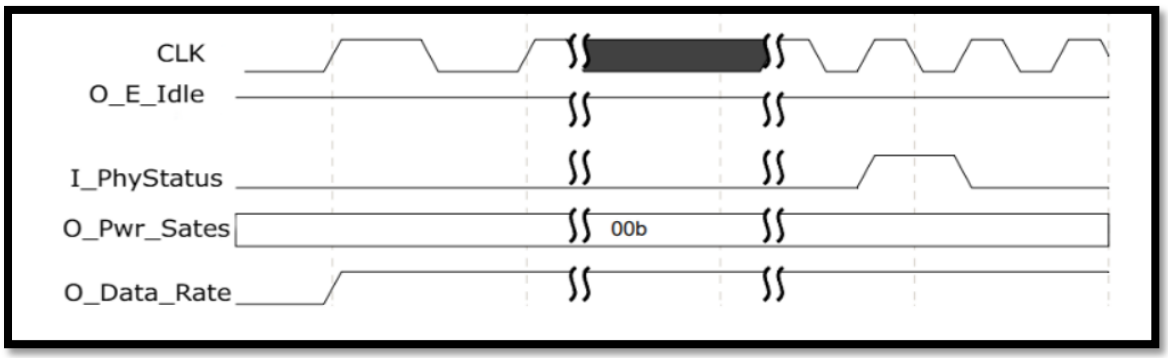
3.5 Electrical Idle

- In many states in the LTSSM, there is a need to forces the transmitter into electrical idle, as there is no data transmission, hence, the O_I_Idle signal is asserted.



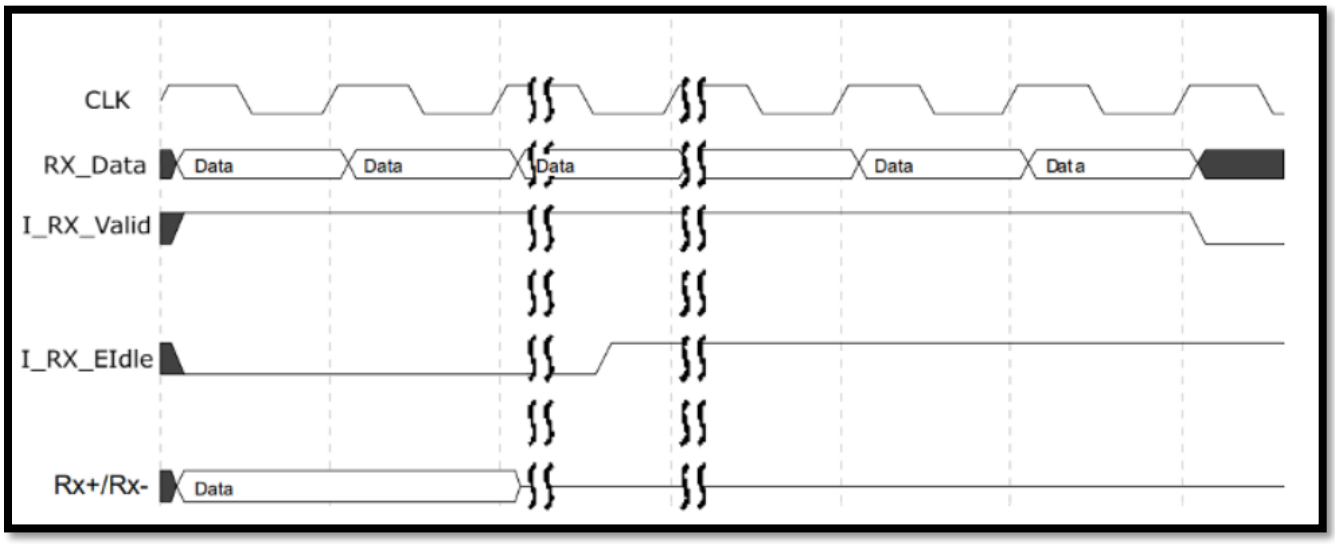
3.6 Data Rate Change

- One of the main roles of the LTSSM is to change the rate of the data when needed, and the pipe must be informed with the desired data rate using the O_Data_Rate signal.



3.7 Detection of Electrical Idle

- In some cases, the LTSSM needs to know the state of the receiver to take an action, as a result there is a signal (I_RX_Idle) that is asserted from the PIPE, when the receiver is detected to be in electrical idle.



❖Design Team Verification input

- **Test case** : description of what the check point is and the aim of this proposed test
- **Test Scenario** : What signals, along with their values, when seen indicates the occurrence of this specific case. In most cases, these signals are Tx_Buffer outputs.
- **Expected** : the expected behavior as described through signals. In most cases, these signals are the Tx_Framing outputs since it's the block that determines which data is to be seen on all lanes in the end..

1. Tx Test Plan

Test case	Test scenario	Expected (Gen 1,2)	Expected (Gen 3 or higher)
Reset	Reset goes low	Throttle = 0 -O_data = 0 -O_Empty = 1 -O_Full = 0 -O_valid = 0	Throttle = 0 -O_data = 0 -O_Empty = 1 -O_Full = 0 -O_valid = 0
1- L0 = 0 During Training i.e: polling_active	Link_Up = 0, while sending OS packets during link training at the beginning.	MAC will continue to send OS packets using the GEN 1 path. i.e: Sending TS1s unframed and unscrambled. o_OS = o_Framed_data = Sc_Data_Out	-

		o_D_K = 32'hFFFF_FFFF	
1-L0 = 0 to L0 = 1 **2-After finishing initial link training or finishing the recovery process sequence and No data is coming from data link layer.**	-Os_enable = 0 -O_Empty = 1 -Link_up = 1	Keep sending Logical Idles. o_D_K = 32'h0	Keep sending Logical Idles.
1-L0 = 1 (Link_UP) **2-one ready packet in Tx buffer** **2-a small-sized TLP(size is less than 28 bytes(i.e 16 bytes from Transaction Layer)** **3-testing the ability of inserting pads when there is no more data.** **4-Beginning of a block**	SERDES = 0 (PIPE Architecture) Symbol_Count = 0 TX_Buffer Outputs -O_sop = 1 -O_Last_Byte= 21 -O_End_valid = 1 -O_type= 1 -O_Length = 4 LTSSM Output -Os_enable = 0	-	-TLP is framed with Start token and pads -no end character -STP (32'b0110_1111_x000_0000_xxxx_ xxxx_ xxxx_ xxxx) +TLP (20 bytes) +8 Idles(64'h0) on all other lanes. -Data will be scrambled. -Tx_Sync_Header = 2'b01 -TX_Start_Block = 1'b1 -valid_data = 1'b1

1-L0 = 1 (Link_UP) **2-one ready packet in Tx buffer** **2-a small-sized TLP(size is less than 28 bytes(i.e 16 bytes from Transaction Layer)** **3-testing the ability of inserting pads when there is no more data.** **4-Adding the sync header at the beginning of a block**	SERDES = 1 (SERDES Architecture) Symbol_Count = 0 TX_Buffer Outputs -O_sop = 1 -O_Last_Byte= 21 -O_End_valid = 1 -O_type= 1 -O_Length = 4 LTSSM outputs -Os_enable = 0	-	-TLP is framed with Start token and pads -no end character -STP (32'b0110_1111_x000_0000_xxxx_ xxxx_ xxxx_ xxxx) +TLP (20 bytes) +8 Idles(64'h0) on all other lanes. -Data will be scrambled. -Sync header (2'b01) will be added. -valid_data = 1'b1
1-L0 = 1 **2-sendig a large sized TLP(greater than 30 bytes ,i.e 56 bytes)** **3-testing sending a new packet without ending it in the same cycle.** **4-Mid Block**	SERDES = 0 (PIPE Architecture) -Os_enable = 0 TX_Buffer Outputs 1st cycle: -O_sop = 1 -O_Last_Byte= 31 -O_End_valid = 0 -O_type = 1 -O_Length = 14 TX_Buffer Outputs 2nd cycle:	-	-TLP is sent over 2 cycles remaining bytes(4) are buffered 1st cycle: -STP (32'b0000_1111_x000_0001_xxxx_ xxxx_ xxxx_ xxxx) +TLP(28). 2nd cycle: -TLP (4+28 bytes). -Data will be scrambled. -Tx_Sync_Header = 2'b01 -TX_Start_Block = 1'b0 -valid_data = 1'b1

	-O_sop = 0 -O_Last_Byte= 29 -O_End_valid = 1 -O_type = 1 -O_Length = 14		
1-L0 = 1 **2-sendig a large sized TLP(greater than 30 bytes ,i.e 56 bytes)** **3-testing sending a new packet without ending it in the same cycle.** **4-Mid Block**	SERDES = 1 (SERDES Architecture) -Os_enable = 0 TX_Buffer Outputs 1st cycle: -O_sop = 1 -O_Last_Byte= 31 -O_End_valid = 0 -O_type = 1 -O_Length = 14 TX_Buffer Outputs 2nd cycle: -O_sop = 0 -O_Last_Byte= 29 -O_End_valid = 1 -O_type = 1 -O_Length = 14	-	-TLP is sent over 2 cycles remaining bytes(4) are buffered 1st cycle: -STP (32'b0000_1111_x000_0001_xxxx_ xxxx_ xxxx_ xxxx) +TLP(28). 2nd cycle: -TLP (4+28 bytes). -Data will be scrambled. -Output of sync logic will be a mixture of the new data and old registered one. -valid_data = 1'b1
-L0 = 1 **2- a ready TLP OR DLLP to be started and an OS should be sent** **3-normal operation &&**	-Os_enable = 1 -O_Empty = 0 -O_sop = 1	-	Packet could be sent if it is not already the end of the current data block, and the packet could be fitted in the currently running block, otherwise idles will be sent till the end of the block

no leftover bytes buffered in framing logic **4-testing priority of OSs**			Then EDS token ends the current data stream After that the OS could be sent
-L0 = 1 **2-aTLP OR DLLP is already in progress and an OS should be sent** **3-testing priority of OSs**	-Os_enable = 1 -O_Empty = 0 -O_sop = 0	-	Packet will be continued till its end, if it ended mid-block, and there is another packet that could be fitted in the remaining of the block, it will be sent, otherwise idles will be added until symbol 15 is reached Then EDS token ends the current data stream After that the OS could be sent
1-L0 = 1 **2-No data is coming from data link layer and no OS is scheduled**	-Os_enable = 0 -O_Empty = 1		Logical idle should be sent on all lanes
1-L0 = 1 **2-testing back pressure due to 128b/130b encoding**	SERDES =1 (SERDES Architecture) -Os_enable = x -O_Empty = x -back_pressure = 1	----------	-Mac will send the accumulated data stored in sync buffer logic and all the logic behind will be in the same state for another one cycle . -R_Enable = 0 -valid_data = 1'b1
1-L0 = 1 **2-testing back pressure due**	SERDES =0 (PIPE Architecture)		-The encoding Process is handled in the PIPE, not by the MAC layer.

to 128b/130b encoding	-Os_enable = x -O_Empty = x -back_pressure = 1	----------	-R_Enable = 0 -valid_data = 1'b0

2. RX test plan

Test case	Test scenario	Expected GEN1,2	Expected Higher Gen's
Reset	Reset at any time	Empty_Rx = 1 All output sympols = 0	Empty_Rx = 1 All output sympols = 0
Handling all different ordered sets	Any received ordered sets such as : TS1'S, TS2'S, SKP	All received ordered set will be forwarded to LTSSM	All received ordered set will be forwarded to LTSSM
L0 = 1 Handling TLP Beginning of block	SERDES = 0 (PIPE Architecture) Handling TLP is received as (STP + TLP + pads) after reset with size is less than 30 bytes (i.e 24 bytes) -RX_Sync_Header = 2'b01 -RX_Start_Block = 1'b1 -RX_Data_Valid = 1'b1	-	- Data will be De-scrambled. -The received TLP Should be forwarded over 1 cycle with: O_sop = 1; O_Last_Byte= 21; O_End_valid = 1 O_Length = 4 Physical_recovery = 0; O_Empty = 0; And idle lanes and framing will be filtered out
L0 = 1 Handling TLP	SERDES = 1 (SERDES Architecture)	-	-Block Alignment will remove the sync header

Beginning of block	Handling TLP is received as (STP + TLP + END + pads) after reset with size is less than 30 bytes (i.e 16 bytes Transaction Layer packet size)		- Data will be De-scrambled. -The received TLP Should be forwarded over 1 cycle with: O_sop = 1; O_Last_Byte= 21; O_End_valid = 1 O_Length = 4 Physical_recovery = 0; O_Empty = 0; And idle lanes and framing will be filtered out
L0 = 1 Handling TLP Mid_Block	SERDES = 0 (PIPE Architecture) Handling TLP is received as (STP + TLP) (32 bytes) -RX_Sync_Header = 2'b01 -RX_Start_Block = 1'b0 -RX_Data_Valid = 1'b1	-	- Data will be De-scrambled. -The received TLP packet should be forwarded over 1 cycle to DLL with: O_sop = 1 O_Last_Byte= 29 O_End_valid = 1 O_Length = O_Empty = 0 O_Rx_Buff_valid_2 =0 -Framing will be filtered out
L0 = 1 Handling TLP	Handling TLP after reset with size is greater than 30 bytes (i.e 64 bytes) TLP are received as : 1st cycle : STP+ TLP 2nd cycle : TLP	-	- Data will be De-scrambled. -The received TLP packet should be forwarded over 2 cycles to DLL with: For 1st cycle O_Rx_Buff_W_EN = 0 For 2nd cycle

	-RX_Sync_Header = 2'b01 -RX_Start_Block = 1'b0 -RX_Data_Valid = 1'b1		O_Rx_Buff_W_EN = 1 O_Rx_Buff_valid_2 =1 O_Empty = 0 O_sop_1 = 1 O_Last_Byte_1= 31 O_End_valid_1 = 0 O_Length_1 = 14 O_sop_2 = 0; O_Last_Byte_2= 29; O_End_valid_2 = 1; O_Length_2 = 14
L0 = 1 Handling logical idle	Any received logical idle characters on all lanes	-	All logical idle will be ignored or filtered by packet filtering
Frame error	SERDES = x (SERDES or PIPE Architecture) existence of Start on unexpected lane number, reception of SOS without EDS before it	-	Physical_recovery = 1
Block Alignment error	SERDES = 1 (SERDES Architecture) Unexpected Sync header (i.e: 2'b00)	-	Physical_recovery = 1
Lane_to_lane Deskew error	SERDES = 1 (SERDES Architecture) one of the lanes is delayed more than 6 cycles than any other lane		Physical_recovery = 1
SKP Removal	SERDES = 1 (SERDES Architecture)		-SKP_remv_rqst = 1 -cnt_en =1 (to increment deleted count)

	Difference between rptr and wptr of elastic buffer is greater than a threshold, and the current symbol is SKP. -ptr_diff > THRESHOLD -isSKP =1	-	-write_en = 0
L0 = 1 Handling reception of SOS	SERDES = x (SERDES or PIPE Architecture)		-Descrambler state will not advance. -Packet Filter will stop processing till end of SOS.
Block Alignment	SERDES =1 (SERDES Architecture) EIEOS is sent, -rx_data = 8'h0 and the EIEOS is monitored correctly		-Transition to monitor_EIEOS -then Transition to aligned phase state for sync_removal -then Transition to waiting for SDS
Block Alignment	SERDES =1 (SERDES Architecture) SDS is sent -rx_data = 8'hE1 then -rx_data = 8'h87 for 14 more symbols		-Transition to locked_phase sync for sync_header_removal -then Transition to locked_phase_data

3. LTSSM test plan

Test case	Test scenario	Expected
Reset -> Detect	• Wait for Clock stabilization.	• Transition to Detect. • Time out of Detect is fired. start = 'b1 time_value1 = 12 ms
Detect_quiet -> Detect_active	• Any lane of Receiver lanes exit electrical idle. l_Rx_EIdle != {‘d32{8’d1}} OR • Time out.	• Transition to Detect_active • All internal variables are cleared.(i.e Link_Up , Directed_speed_change).
Detect_active -> Pre_Polling	• All Lanes have detected a receiver. l_Rcv_detected = {‘d32{8’d1}}	• O_st_detect = {‘d32{8’d1}} • Transition to Pre-Polling.
Detect_active -> Detect again	• Not all lane has detected a receiver. l_Rcv_detected != {‘d32{8’d0}}	• Transition to Detect again.
Detect_again: Restart receiver detection only after 12 ms time out.	• After 12ms are timed out	• O_st_detect = {‘d32{8’d1}}
Detect_again -> Pre_Polling	• All previously detected Lanes have detected a receiver again	• Transition to Pre_Polling
Pre_Polling -> Polling Active	• Wait for TX Exit from Electrical Idle. (i_phy_status = 1). • And operate at normal mode	• Transition to Polling_Active • controller_rst = 1'b1 reset_ack = 1'b1
Polling_Active -> Polling_Config	• All detected lanes receive 8 consecutive TS1s.	• Transition to Polling_Config • os_enable = 1 controller_rst = 1'b1

	• All detected lanes Transmitted 1024 TS1s. • Exit_state != 0. , ack = 1	reset_ack = 1'b1
Polling_Config -> Config_Linkwidth Start	• any detected lane and lane 0 should be included receive 8 consecutive TS2s • Any detected lane Transmitted 16 TS2 after receiving 1. • Exit_state != 0. , ack = 1	• Transition to Config_Linkwidth_Start • os_enable = 1 controller_rst = 1'b1 reset_ack = 1'b1
Config_Linkwidth _Start -> Config_Linkwidth _Accept	• Any detected lane receive 2 consecutive TS1s with non-PAD link number and PAD lane numbers with de-asserted loopback and disable bits. • Exit_state != 0.	• Transition to Config_Linkwidth_Accept • os_enable = 1 controller_rst = 1'b1 reset_ack = 1'b1
Config_Linkwidth _Accept -> Config_lanenum_ Wait	• All lanes that transmitted TS1s with non-PAD link number receive 2 TS1s with non-PAD link and lane numbers. • Exit_state = non_PAD_Link_Lanes.	• Transition to Config_Lanenum_wait. • os_enable = 1 controller_rst = 1'b1 reset_ack = 1'b1
Config_Lanenum_ Wait -> Config_Lanenum_ Accept	• All lanes that transmitted TS1s with non-PAD link and lane numbers receive 2 TS2s with non-PAD link and lane numbers. OR • All lanes that transmitted TS1s with non-PAD link and lane numbers receive 2 TS1s with different non-PAD link lane numbers from that transmitted. • Exit_state = non_PAD_Link_Lanes.	• Transition to Config_Lanenum_Accept • os_enable = 1 controller_rst = 1'b1 reset_ack = 1'b1

Config_Lanenum_ Accept -> Config_Lanenum_ Wait	• All lanes that transmitted TS1s with non-PAD link and lane numbers receive 2 TS1s with non-PAD link and lane numbers. • Exit_state = non_PAD_Link_Lanes.	• Transition to Config_Lanenum_Wait • controller_rst = 1'b1 os_enable = 1 reset_ack = 1'b1
Config_Lanenum_ Accept -> Config_Complete	• All lanes that transmitted TS1s with non-PAD link and lane numbers receive 2 TS1s with non-PAD link and lane numbers. • Exit_state = non_PAD_Link_Lanes.	• Transition to Config_Complete. • os_enable = 1 controller_rst = 1'b1 reset_ack = 1'b1
Config_Complete- >Config_idle_2	• All lanes that transmitted TS1s with non-PAD link and lane numbers receive 8 TS2s with non-PAD link and lane numbers. • Transmit 16 TS2s after transmit 1. • Exit_state = non_PAD_Link_Lanes. ,, ack = 1. • o_Config_lanes = 0 • GEN = LOW_GEN	• Transition to Config_idle_2 • os_enable = 1 Other lanes than lane 0 are powered down. controller_rst = 1'b1 reset_ack = 1'b1
Config_Complete- >Config_idle	• All lanes that transmitted TS1s with non-PAD link and lane numbers receive 8 TS2s with non-PAD link and lane numbers. • Transmit 16 TS2s after transmit 1. • Exit_state = non_PAD_Link_Lanes. ,, ack = 1. • o_Config_lanes = 0 • GEN = HIGH_GEN	• Transition to Config_idlE • os_enable = 1 Power = p1 (for lanes other than 0) controller_rst = 1'b1 reset_ack = 1'b1
Config_idle -> Config_idle_2	• Transmitter must send an SDS	• Transition to Config_idle_2 • os_enable = 1

		controller_rst = 1'b1 reset_ack = 1'b1
Config_idle_2 -> L0	• All Configured lanes receive 8 idles. • Transmit 16 idles after receiving one idle. • Exit_state = configured lanes • GEN = LOW_GEN OR • GEN = HIGH_GEN, and this state was not entered from config_complete due to timeout.	• Transition to L0 • os_enable = 0 controller_rst = 1'b1 reset_ack = 1'b1
L0 -> recovery_rcvrlock	• Reached L0 with Low GEN and higher data rates are supported • An order set is received in L0 (indicating that the other device has initiated the recovery process) (speed change) • Type_IDL_TS = 1	• Transition to recovery_rcvrlock • os_enable = 0 controller_rst = 1'b1 reset_ack = 1'b1
L0 -> pre_recovery	• Reached L0 with Low GEN and higher data rates are supported • Idles are received, the device is considered the initiator of the speed change process. • Type_IDL_TS = 0	• Transition to pre_recovery • os_enable = 0 directed_speed_change= 1 controller_rst = 1'b1 reset_ack = 1'b1
pre_recovery-> recovery_rcvrlock	• GEN = LOW GEN ▪ Trasmit TS1s with speed bit asserted. ▪ Wait for COM reception, indicating that the other device has entered recovery as well. ▪ Type_IDL_TS = 1 OR	• Transition to recovery_rcvrlock • os_enable = 1 controller_rst = 1'b1 reset_ack = 1'b1

	• GEN = HIGH GEN ▪ Trasmit TS1s with speed bit asserted. ▪ Wait for COM reception, indicating that the other device has entered recovery as well.	
recovery_rcvrlock -> recovery_rcvrconfig	• Transmit TS1s with speed bit equals to directed_speed_change variable • All configured lanes receive 8 TS1s speed bit equals to the directed_speed_change variable recovr_speedequ_config = 0 OR • Timeout and All configured lanes receive 8 TS1s speed bit equals 1 recovr_recvrconfg = non_pad_link_lanes	• Transition to recovery_rcvrconfig • os_enable = 1 controller_rst = 1'b1 reset_ack = 1'b1
recovery_rcvrconfig-> recovery_idle_skp	• Transmit TS2s with speed bit equals to directed_speed_change variable • Any configured Lane receives 8 TS2s with speed bit de-asserted.	• Transition to recovery_idle_skp • controller_rst = 1'b1 reset_ack = 1'b1 directed_speed_change =0 changed_speed_recovery =0 successful_speed_negotiation = 1
recovery_rcvrconfig-> recovery_speed_EIOS	• Transmit TS2s with speed bit equals to directed_speed_change variable • Any configured Lane receives 8 TS2s with speed bit asserted.	• Transition to recovery_speed_EIOS • controller_rst = 1'b1 reset_ack = 1'b1 successful_speed_negotiation = 1

recovery_rcvrconfig-> config_Linkwidth_start	• Transmit TS2s with speed bit equals to directed_speed_change variable • Any configured Lane receives 8 TS1s with different link and lane numbers than those registered since leaving configuration	• Transition to Config_Linkwidth_start • controller_rst = 1'b1 reset_ack = 1'b1
recovery_speed_EIOS-> recovery_speed_TX_IDLE	• Transmit one EIOS • ack = 1	• Transition to recovery_speed_TX_IDLE • os_enable = 1 controller_rst = 1'b1 reset_ack = 1'b1
recovery_speed_TX_IDLE -> recovery_speed_RX_IDLE	• Bring the transmitter power down (power = p1) • Wait for the power down process completion (I_Phystatus = 1) • The receiver is not idle yet I_RX_Idle != {‘d32{1’b1}}	• Transition to recovery_speed_RX_IDLE
recovery_speed_TX_IDLE -> recovery_speed_rate_change	• Bring the transmitter power down (power = p1) • Wait for the power down process completion (I_Phystatus = 1) • The receiver lanes have entered electrical idle • I_RX_Idle = {‘d32{1’b1}}	• Transition to recovery_speed_rate_change • start_speed_neg = 1 (indicating the 800 ns timer)
recovery_speed_RX_IDLE -> recovery_speed_rate_change	• The receiver lanes have entered electrical idle • I_RX_Idle = {‘d32{1’b1}}	• Transition to recovery_speed_rate_change • start_speed_neg = 1 (indicating the 800 ns timer)
recovery_speed_rate_change -> recovery_speed_power_up	• Change the rate (o_rate = 3’b100) (32GT/s) • Wait for the assertion of I_Phystatus • 800 ns have already passed since entering	• Transition to recovery_speed_power_up

	recovery_speed_rate_change	
recovery_speed_rate_change -> recovery_speed_wait_neg	• Change the rate (o_rate = 3'b100) (32GT/s) • Wait for the assertion of I_Phystatus The 800 ns specified by the standard did not pass yet	• Transition to recovery_speed_wait_neg
recovery_speed_wait_neg -> recovery_speed_power_up	• Wait for the 800 ns specified by the standard • Timeout3 = 1	• Transition to recovery_speed_power_up
recovery_speed_power_up -> recovery_speed_EIEOS	• Bring the power back up (power = p0) • Wait for the process completion (I_Phystatus = 1)	• Transition to recovery_speed_EIEOS
recovery_speed_EIEOS -> recovery_speed_recovery_rcvrlock	• Transmit 16 EIEOS • ack = 1 os_creator_done = 1	• Transition to recovery_speed_recovery_rcvrlock
Recovery_idle_skp -> Recovey_idle_sds	• Transmit 1 Ctrl SKP Os_creator_done = 1	• Transition to Recovery_idle_sds • os_enable = 1 controller_rst = 1 reset_ack = 1
Recovery_idle_idl -> L0	• Receive 8 idles Exit_state != 0 • Transmit 16 idles ack_idle = 1	• Transition to L0 • os_enable = 0 PF_EN = 1, Controller_rst = 1, Link_up = 1, o_Retrain_succ = 1 •
Recovery_idle_sds -> Recovey_idle_idl	• Transmit 1 sds Os_creator_done = 1	• Transition to Recovery_idle_idl • os_enable = 0, controller_rst = 1 reset_ack = 1 •
L0 -> Pre_Recovery	• Rx_error = 1,	• physical_recovery = 1. • Transition to Pre_Recovery.

(receiver_error)		
L0 -> Pre_Recovery (Data link Layer wants to retrain the link)	• Retrain = 1,	• physical_recovery = 1. • Transition to Pre_Recovery.