SRIO VIP Microarchitecture

V1.7

NOTICE

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE DOCUMENT. MOBIVEIL, INC. OR IT’s LICENSOR MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE TECHNOLOGY DESCRIBED IN THIS DOCUMENT AT ANY TIME.

Revision History

Revision	Date	By	Change
1.0	03/01/2013	MV	Initial Version
1.1	03/27/2013	MV	Incorporated the feedback from 03/06/2013 and 03/21/2013 meetings.
1.2	04/02/2013	MV	Updated based on the comments received from 03/28/2013 conference call.
1.3	04/10/2013	MV	Incorporated the feedback from 04/04/2013 meeting.
1.4	04/26/2013	MV	Few more config variables and checks are included based on the review feedback.
1.5	06/03/2013	MV	Updated based on the comments from RTA members and included variables added for 1.1 PATCH1 release.
1.6	07/26/2013	MV	GEN3 Changes
1.7	09/20/2013	MV	GEN3 Changes

Glossary

SRIO	Serial RapidIO
VIP	Verification IP
BFM	Bus Functional Model
VC	Virtual Channel
GSM	Globally Shared Memory
LFC	Logical Flow Control
PL	Physical Layer
TL	Transport Layer
LL	Logical Layer
ENV	Environment
REG	Register
DUT	Design Under Test
TLM	Transaction Level Model
FC	Functional Coverage
I/O	Input/Output
DS	Data Streaming
CS	Control Symbol
VMIN	Minimum Valid Characters
UVM	Universal Verification Methodology
SV	System Verilog
CRF	Critical Request Flow
ISM	Initialization State Machine
AET	Adaptive Equalization Training

Contents

Figures

Tables

1 Introduction

1.1 Purpose

This document provides the description of SRIO VIP components, Sequences and the protocol checks.

1.2 Scope

The document covers the implementation details of SRIO VIP components and the identified sequences.

1.3 Audience

This document is intended for test bench developers and testcase writers who is going to use the SRIO VIP.

1.4 References

- *"Serial RapidIO specification 1.3"*
- *"Serial RapidIO specification 2.x"*
- *"Serial RapidIO specification 3.x"*
- *"UVM 1.1 class reference manual"*
- *"UVM user guide 1.1"*

2 Overview

Mobiveil's Gen3 SRIO VIP supports SRIO specification versions 3.0, 2.2,2.1 and 1.3. The Gen3 VIP is system verilog (SV) based and supports standard Universal Verification Methodology (UVM). Mobiveil's Gen3 SRIO VIP can be easily plugged in to any other UVM compliant verification components to extend a broader verification environment.

2.1 Features Supported

- Supports Serial RapidIO specification versions 3.0, 2.2, 2.1 and 1.3
- Supports 1x, 2x, 4x 8x and 16x lane configurations.1.25 Gbaud, 2.5 Gbaud, 3.125 Gbaud, 5 Gbaud, 6.25 and 10.3125 Gbaud lane rates
- Supports 66, 50 and 34-bit addressing on the RapidIO interface
- Supports all types of packet formats
- Supports all types of IDLE sequences, Control and Status Symbols
- Supports Scrambling/De-Scrambling and Encoding/Decoding
- Supports out of order transaction generation and handling
- Supports critical request flow (CRF)
- Supports all transaction flows, with all priorities
- Supports test pattern generation at all protocol layers
- Supports error injection and error detection at all levels of protocol layers
- Provides Compliance Test Suite
- Functional Coverage

3 SRIO VIP Architecture

Mobiveil's Gen3 SRIO VIP uses layered architecture which is divided in to a Logical, Transport and Physical layer. RapidIO monitor handles protocol checking and complies with Serial RapidIO specification. It provides hooks for implementing functional coverage, scoreboard and checker.

The Serial RapidIO VIP provides extensive compliance test suite and reduces the verification effort. It can be used to verify IP, SoC and system level designs. Stimulus generation is automated and gives large flexibly for the user to generate directed and random test scenarios.

The user can constrain the randomization at different levels and functional coverage helps gauging the effectiveness of the randomization. Detailed description of each Sequence is provided and allows the user to extend and implement the required functionality.

SRIO VIP includes register model which models the registers defined in the Serial RapidIO specification and provides coverage of these registers. Register model is used to verify the properties of the registers. Additionally it is used by the scoreboard, monitor and Functional coverage models.

SRIO collects and reports functional coverage data. It provides detailed report log on the packets and symbols. Multiple levels of verbosity is supported which makes the debugging easy.

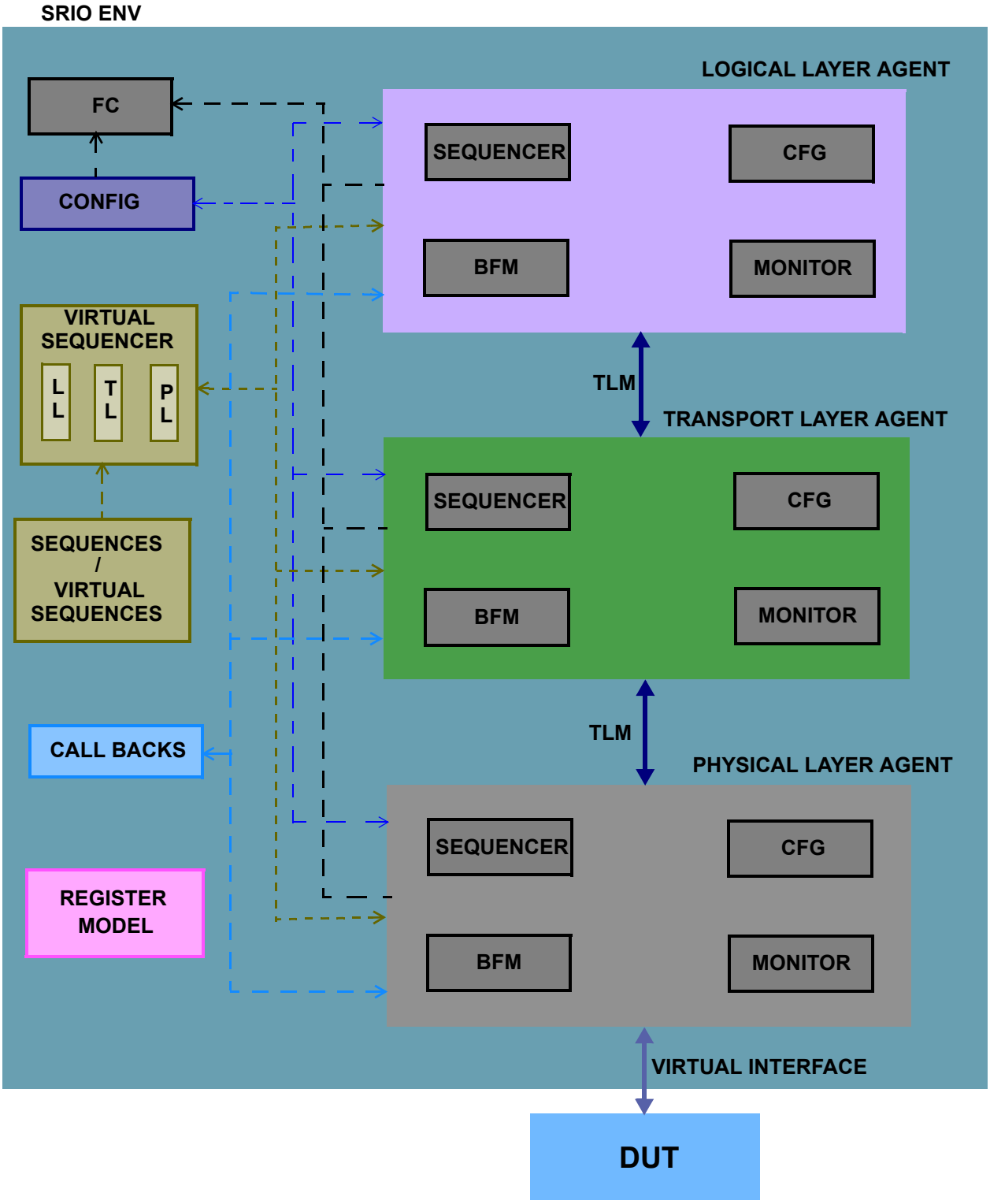
Mobiveil's SRIO VIP can be easily integrated in to cycle-based or event-based simulation environments and can support latest versions of major simulation tools in the industry.

Figure 1, "Architecture of the Mobiveil's GEN3 SRIO VIP," on page 12 shows block diagram and top level architectural details.

The top unit that instantiates all other units in this architecture is srio_env.srio_env is inherited from uvm_env. Within in the srio_env, the following components are instantiated.

- Config
- Register Model
- Logical Layer Agent
- Transport Layer Agent
- Physical Layer Agent
- Sequences
- Callbacks

Figure 1 : Architecture of the Mobiveil's GEN3 SRIO VIP



3.1 Configuration

This is the configuration object module that contains the global configuration variables used by the SRIO VIP components.This class inherits from uvm_object and named as srio_config.

Refer to Table 1, “SRIO VIP Global Configuration Parameters,” on page 13 for the list of global configuration variables and their description.

Table 1: SRIO VIP Global Configuration Parameters

S.No	Parameter	Description
1	is_active	UVM_ACTIVE - BFM component will be instantiated and drive the interface signals. UVM_PASSIVE - Only Monitor component will be instantiated.Default - UVM_ACTIVE.
2	srio_vip_model	Selects the sRIO VIP model.(PE Model - SRIO_PE,PL Model - SRIO_PL,TxRx Model - SRIO_TXRX). Default - SRIO_PE.
3	srio_mode	Configures the mode/version. (SRIO_GEN30,SRIO_GEN22,SRIO_GEN21 and SRIO_GEN13). Default - SRIO_GEN21.
4	num_of_lanes	Configures the number of lanes.(1,2,4,8 and 16).Default - 4.
5	srio_baud_rate	Selects the baud rate. (SRIO_125,SRIO_25,SRIO_3125,SRIO_5, SRIO_625 and SRIO_103125).
6	en_ext_addr_support	If value is “1”, supports extended address. Default - 0.
7	srio_addr_mode	Configures the addressing mode.(SRIO_ADDR_34,SRIO_ADDR_50 and SRIO_ADDR_66).Default - SRIO_ADDR_34.
8	srio_dev_id_size	Configures the device id type.(SRIO_DEVID_8,SRIO_DEVID_16 and SRIO_DEVID_32).Default - SRIO_DEVID_8.

S.No	Parameter	Description
9	srio_interface_mode	Configures if the line interface is serial or parallel.(SRIO_SERIAL, SRIO_PARALLEL).If Serial interface is selected, data is driven serially and user needs to provide serial clock.If Parallel interface is selected, data is driven in 10-bit or 67-bit interfaces based on GEN1/2 and GEN3 mode, user needs to provide parallel clock.Default - SRIO_SERIAL.
10	reg_space_size	Configures the size of the register block.Default - 24'hFFFFFF.
11	srio_reg_model_tx	Pointer for the BFM's srio register model instance.
12	srio_reg_model_rx	Pointer for the DUT's srio register model instance.
13	link_initialized	Informs the status of the link initialization of active component.
14	pl_mon_tx_link_initialized	Informs the status of the link initialization of TX monitor.
15	pl_mon_rx_link_initialized	Informs the status of the link initialization of RX monitor.
16	packet_rx_started	Event which is triggered when the BFM started receiving a packet.
17	current_ack_id	Provides the value of the AckID currently used.
18	srio_tx_mon_if	Configures the TX monitor type.(BFM or DUT).
19	srio_rx_mon_if	Configures the RX monitor type.(BFM or DUT).
20	pl_rx_mon_init_sm_state	Provides the ISM state of RX monitor.
21	pl_rx_mon_init_sm_state	Provides the ISM state of TX monitor.
22	idle_detected	Informs that active component has completed the IDLE sequence detection
23	idle_selected	Informs the IDLE sequence detected by the active component. 1 indicates IDLE2, 0 indicates IDLE1.
24	multi_vc_support	Multiple VC Support.When true, VC1-8 and VC0 supported. When false, VC0 support. Default - 0.

S.No	Parameter	Description
25	vc_num_support	VC Number Support. Number of VC's supported.Default - 1.
26	file_h	Integer variable for file handler. Used for tracker file generation.
27	ll_config	Handle of srio_ll_config.
28	tl_config	Handle of srio_tl_config.
29	pl_config	Handle of srio_pl_config.
30	port_number	Device Port Number.Default - 0.
31	spec_support	Indicates the specification version to be followed. It is used to enable the new features added for 1.3 and 2.x devices in 3.0 specification, like timing control symbols, link-request reset port command, and new register set etc.
32	en_packet_delay	Enables delay between packets. Default - 0.

3.2 Register Model

The Register Model (srio_reg_model) contains all the registers defined in the Serial RapidIO specifications.These registers act as mirror to the DUT registers and implemented using UVM register package. All the registers are implemented in little-endian format.

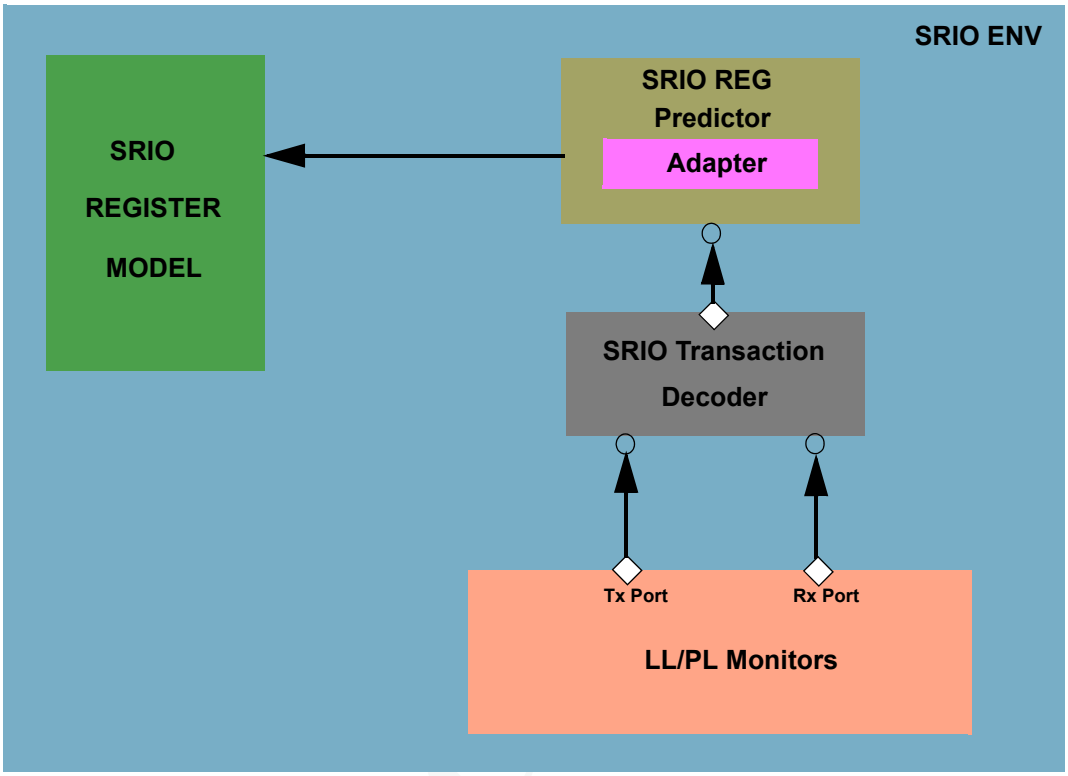
The SRIO Register Layer is constructed using the below components.

- SRIO Register Model
- SRIO Transaction Decoder
- SRIO Register Adapter
- SRIO Register Predictor

All the read and write accesses to the registers of SRIO DUT are monitored by the register layer and necessary updation is done to the register model. The register layer implemented here uses passive prediction and hence the register model takes no active part in any accesses to the DUT, but it is kept up to date when any front door register accesses take place.

Figure 2, "Block Diagram of SRIO Register Layer," on page 16 shows the block diagram of SRIO Register Layer.

Figure 2 : Block Diagram of SRIO Register Layer



3.2.1 SRIO Register Model

The Register Model contains all the SRIO registers in little-endian format and its corresponding address map.

3.2.2 SRIO Transaction Decoder

SRIO Transaction Decoder receives the srio_trans sequence item from the LL or PL agent's monitor component and passes only the transactions that can access SRIO registers (Maintenance and NWRITE_R or NREAD packets) to SRIO register predictor module. The received Maintenance, NWRITE_R or NREAD transactions are queued and sent one by one to SRIO Register Predictor module, after receiving DONE response for that particular transaction.

If the VIP is configured as PE model, the input ports of SRIO transaction decoder are connected to Tx and Rx analysis ports of LL monitor. If the VIP is configured as PL model, then the input ports are connected to Tx and Rx analysis ports of PL monitor.

3.2.3 SRIO Register Adapter

The SRIO Register Adapter converts SRIO transactions (Maintenance, NWRITE_R/NREAD) into register transactions. The conversion of srio_trans sequence item into a register transaction is implemented inside bus2reg function of Adapter class.

3.2.4 SRIO Register Predictor

Register Predictor block receives the Maintenance, NWRITE_R or NREAD transactions from the SRIO Transaction Decoder block and it calls the bus2reg function of Register Adapter. The output of bus2reg function is a register transaction, which is then used to update the corresponding register in the Register model.

3.3 Virtual Sequencer

SRIO virtual sequencer (srio_virtual_sequencer) contains the references of LL,TL,PL sub sequencers. Virtual sequencer helps in coordinating the time and the data between different agents.Virtual sequences are associated with this virtual sequencer and are used to coordinate stimulus generation in LL,TL and PL agents.

3.4 Logical Layer Agent

This section provides the details of logical layer agent and it's sub components.It also describes how logical layer agent interacts with transport layer agent.

The logical layer lgent (srio_ll_agent) implements logical layer functionalities of the SRIO standard. The logical layer agent performs the function of transmitting and receiving the logical layer data. srio_ll_agent is extended from the UVM base class uvm_agent and its parent component is srio_env.

Logical Layer Agent consists of sub-blocks listed below:

- LL Config
- LL Sequencer
- LL BFM
- LL Monitor

Figure 3, "Block Diagram of Logical Layer Agent," on page 21 shows the Logical Layer Agent.

3.4.1 LL Configuration

Logical Layer configuration object (srio_ll_config) is inherited from uvm_object, has the configuration variables used by the components of logical layer agent.

Refer to Table 2, “Logical Layer Configuration Parameters,” on page 18 for the list of logical layer configuration variables and their description.

Table 2: Logical Layer Configuration Parameters

S.No	Parameters	Description
1	is_active	UVM_ACTIVE - BFM component will be instantiated and drive the interface signals. UVM_PASSIVE - Only Monitor component will be instantiated.Default - UVM_ACTIVE.
2	has_checks	If TRUE, enables the LL checks in the LL monitor else checks are disabled.Default - TRUE.
3	has_coverage	If TRUE, functional coverage for logical layer is enabled.Default - TRUE.
4	interleaved_pkt	If TRUE, packets of different types are interleaved. If FALSE, packets are sent in the order they are received.Default - FALSE.
5	is_active	UVM_ACTIVE - BFM component will be instantiated and drive the interface signals. UVM_PASSIVE - Only Monitor component will be instantiated.Default - UVM_ACTIVE.
6	en_out_of_order_gen	If TRUE, messages and responses are generated out-of-order.Default - FALSE.
7	arb_type	Configures the arbitration mechanism of packet scheduler.(SRIO_LL_RR - Round Robin, SRIO_LL_WRR - Weighted Round Robin).Default - SRIO_LL_RR.
8	resp_done_ratio	Defines the probability of DONE response generation.Default - 100.
9	resp_err_ratio	Defines the probability of ERROR response generation.Default - 0.
10	resp_retry_ratio	Defines the probability of RETRY response generation.Default - 0.
11	gen_resp_en_ratio	Defines the probability of sending the response packet.Default - 100.
12	gen_resp_dis_ratio	Defines the probability of disabling the response packet.Default - 0.
13	resp_interv_ratio	Defines the probability of INTERVENTION response generation.Default - 20.
14	resp_done_interv_ratio	Defines the probability of DONE INTERVENTION response generation.Default - 20.

S.No	Parameters	Description
15	resp_data_only_ratio	Defines the probability of DATA ONLY response generation.Default - 20.
16	resp_not_owner_ratio	Defines the probability of NOT OWNER response generation.Default - 0.
17	resp_gen_mode	IMMEDIATE, RANDOM,DISABLED.Default - IMMEDIATE.
18	resp_delay_min	Configures the response delay minimum value.Default - 100.
19	resp_delay_max	Configures the response delay maximum value.Default - 500.
20	io_pkt_ratio	Defines the probability of IO Packets transmission.Default - 20.
21	msg_pkt_ratio	Defines the probability of Message Packet transmission.Default - 20.
22	db_pkt_ratio	Defines the probability of Doorbell Packet transmission.Default - 20.
23	gsm_pkt_ratio	Defines the probability of GSM Packets transmission.Default - 20.
24	lfc_pkt_ratio	Defines the probability of LFC Packets transmission.Default - 20.
25	ds_pkt_ratio	Defines the probability of Data Streaming Packets transmission.Default - 20.
26	bfm_tx_pkt_cnt	Transmitted packet count from LL BFM
27	bfm_rx_pkt_cnt	Received packet count from LL BFM.
28	block_ll_traffic	Enable/Disable transmission from LL.Default - FALSE.
29	ll_resp_timeout	Response timeout value in nanoseconds.Default - 10000.
30	ll_pkt_transmitted	Event triggered whenever a packet is transmitted from LL.
31	ll_pkt_received	Event triggered whenever a packet is received by LL.
32	orph_xoff_timeout	Timeout limit for Orphaned XOFF in ns. Default - 32'h0FFF_FFFF
33	req_xonxoff_timeout	Max time (in ns) within which XON/XOFF to be sent after receiving REQUEST. Default - 32'h0FFF_FFFF

S.No	Parameters	Description
34	xon_pdu_timeout	Max time (in ns) within which PDU has to be sent after receiving XON Default - 32'h0FFF_FFFF
35	tx_mon_tot_pkt_rcvd	Number of packets received by TX monitor
36	rx_mon_tot_pkt_rcvd	Number of packets received by RX monitor

3.4.2 LL Sequencer

The logical Layer agent sequencer (srio_ll_sequencer) has a sequence driver and a sequence item.The role of the sequencer is to route the sequence item from a sequence to the bfm.The sequence item is of class type srio_trans and the fields are defined in Table 15, “SRIO VIP’s Sequence Item,” on page 78.

3.4.3 LL BFM

The logical Layer BFM handles generation and reception of logical layer packets.The sub-modules involved in the logical layer BFM are listed below.

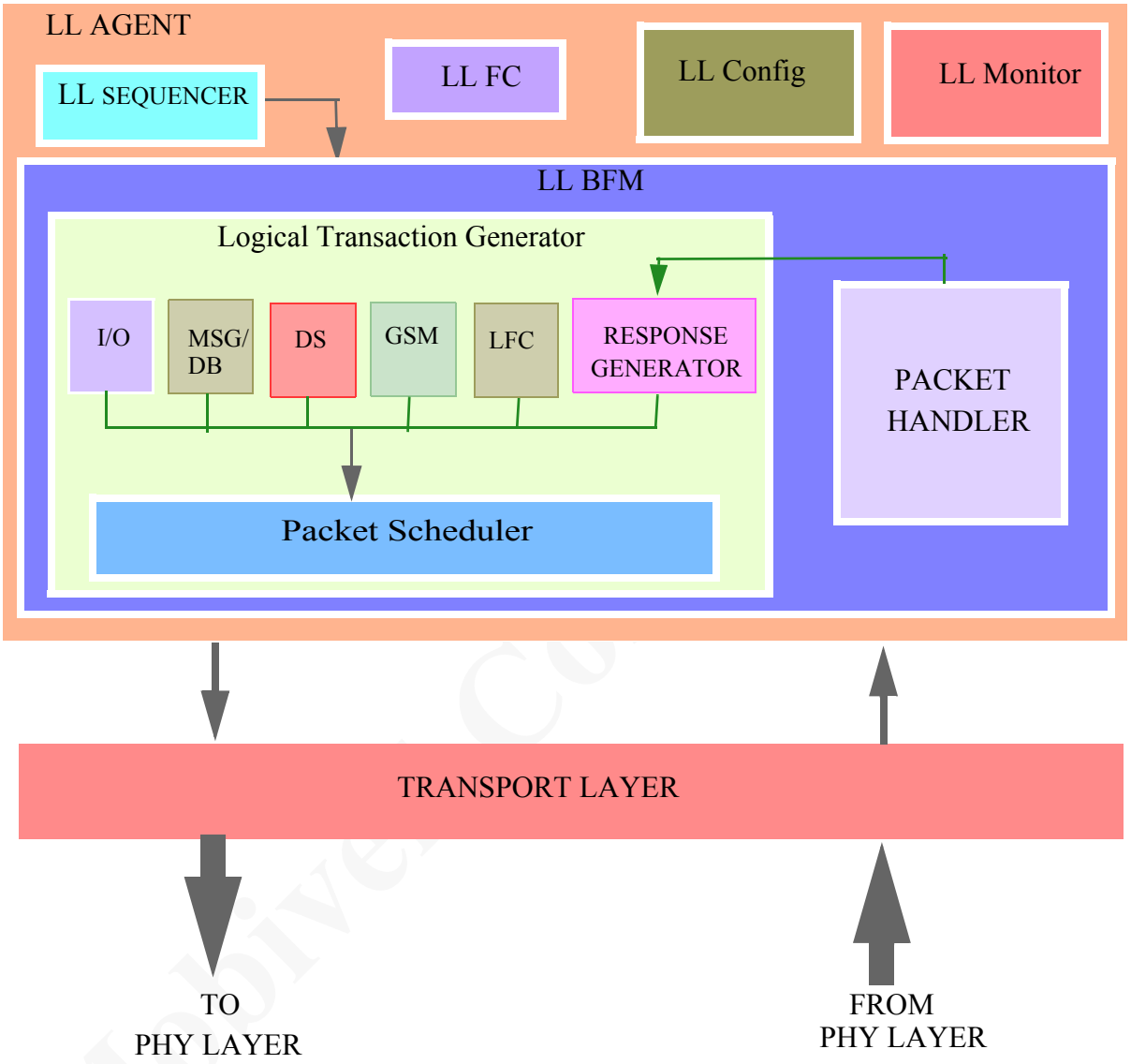
- Logical Transaction Generator
- Packet Handler

3.4.3.1 Logical Transaction Generator

Logical transaction generator handles passing the packets received from LL sequence to individual generators and also reads the packets from individual generators and sends them to TL. When a packet is received from sequence, it is checked for valid ftype and ttype values. Based on the ftype value, packet is sent to the respective generator for further processing.It also implements packet scheduling mechanism. Packets can be transmitted in interleaved or non interleaved mode based on the configuration from user. Also in the interleaved mode, it can be Round Robin or Weighted Round Robin.When packets are available to transmit from generators, reads the packet using the transmission method configured and transmits to TL using TLM port.Transaction Generator instantiates following sub modules.

- I/O
- MSG
- DS
- GSM
- LFC
- Response Generator
- Packet Scheduler

Figure 3 : Block Diagram of Logical Layer Agent



3.4.3.1.1 I/O

I/O module generates required payload for the of IO packets using wdptr and wrsize value.Since valid values for IO fields are constrained in srio_trans, those values are not modified. Also user generated packets are not modified.This module participates in packet scheduling logic and provides the IO packets when available.It also ensures providing non posted transactions only if a Transaction ID is available.The different Packet types generated by this module are shown in Table 3, “I/O Packet Types,” on page 22.

Table 3: I/O Packet Types

Type	Description
NREAD	Read specified address
NWRITE	Write specified address
NWRITE_R	Write specified address, notify source of completion
SWRITE	Write specified address
ATOMIC SET	Read-write 1s to specified address
ATOMIC CLEAR	Read-write 0s to specified address
ATOMIC INCREMENT	Read-increment-write to specified address
ATOMIC DECREMENT	Read-decrement-write to specified address
ATOMIC TEST-AND-SWAP	Read-test=0-swap-write to specified address
ATOMIC SWAP	Read-write to specified address
ATOMIC COMPARE-AND-SWAP	Read-test=first data-write second data to specified address
MAINTANENCE READ	Read device configuration registers and perform other system maintenance tasks
MAINTENANCE WRITE	Write device configuration registers and perform other system maintenance tasks
MAINTENANCE PORT-WRITRE	Maintenance port-write operation

3.4.3.1.2 MSG/DB

MSG/DB module takes care of generating message and door bell packets.(TYPE 10, TYPE 11). The different Packet types generated by this module are listed in Table 4, “Message Packet Types,” on page 22.

User can generate all the segments belong to a message and push it via logical layer sequencer or can instruct this module to generate the segments by passing few specific fields like segment size, message length etc. This module handles the functionality of providing the packets in out of order fashion to packet scheduler. It provides Doorbell packet only when a Transaction ID is available and generates message packets only if the letter/mailbox is not in use.MSG/DB generator handles retry and response timeout mechanism for message and doorbell packets.

Table 4: Message Packet Types

Type	Description
DATA MESSAGE	Send a message
DOORBELL MESSAGE	Send a short message

3.4.3.1.3 DS

DS module generates the data streaming packets (TYPE 9) as per the packet fields received from the sequencer. It also generates Extended type-9 packet as per the packet fields received.

All segments of a data streaming packets can be generated by the user in the sequence or DS module can generate the required segments based on the specific fields provided by the user.

Table 5, “Data Streaming Packet Types,” on page 23 lists the packet types generated by this component.

Table 5: Data Streaming Packet Types

Type	Description
DATA STREAMING	Streaming Packet with xh=0
DATA STREAMING with Extended Header	Streaming Packet with xh=1

3.4.3.1.4 GSM

GSM module generates required payload for the of GSM packets using wdptr and wrsize value.Since valid values for GSM fields are constrained in srio_trans, those values are not modified. Also user generated packets are not modified.This module participates in packet scheduling logic and provides the GSM packets when available.It also ensures providing non posted transactions only if a Transaction ID is available. GSM module handles response retry and response timeout logic for gsm packets.The different packet transaction types generated by this module are listed in Table 6, “GSM Packet Types,” on page 23.

Table 6: GSM Packet Types

Type	Description
READ_OWNER	Read shared copy of remotely owned coherence granule
READ_TO_OWN_OWNER	Read for store of remotely owned coherence granule
IO_READ_OWNER	Read for I/O of remotely owned coherence granule
READ_TO_OWN_HOME	Read for store of home memory for coherence granule
READ_HOME	Read shared copy of home memory for coherence granule
IO_READ_HOME	Read for I/O of home memory for coherence granule
DKILL_HOME	Invalidate to home memory of coherence granule
IKILL_HOME	Invalidate to home memory of coherence granule
TLBIE	Invalidate TLB entry
TLBSYNC	Synchronize TLB invalidates

Type	Description
IREAD_HOME	Read shared copy of home memory for instruction cache
FLUSH	Force return of ownership of coherence granule to home memory, no update to coherence granule
IKILL_SHARER	Invalidate cached copy of coherence granule
DKILL_SHARER	Invalidate cached copy of coherence granule
CASTOUT	Return ownership of coherence granule to home memory
FLUSH (With DATA)	Force return of ownership of coherence granule to home memory, update returned coherence granule

3.4.3.1.5 LFC

LFC module receives fully created LFC packets from sequence->logical transaction generator and participates in packet scheduling mechanism to send those packets to TL. Table 7, “Logical Flow Control Packet Types,” on page 24 shows LFC packet types.

Table 7: Logical Flow Control Packet Types

Type	Description
REQUEST	Resources Allocation
XOFF	Indicate unavailability of resources
XON	Allow and grant use of resources
RELEASE	Resources De-allocation

3.4.3.1.6 Response Generator

Response generator generates response packets for the request packets received by the packet handler that require response. It participates in packet scheduling mechanism and provided the response packets to logical transaction generator. It generates various responses and response delays using the values configured in the LL config. It is capable of providing the responses in out-of-order fashion.

Refer to Table 8, “Response Packet Types,” on page 24 for various types of response packets generated by the response generator.

Table 8: Response Packet Types

Type	Description
DONE	Indicates to the requestor that the desired transaction has completed

Type	Description
RETRY	Message: Mailbox is busy servicing another message operation Doorbell: It encounters busy doorbell hardware. GSM: Address conflicts within the system that need resolution
ERROR	The target of the transaction encountered an unrecoverable error and could not complete the transaction
INTERVENTION	Responses are generated as part of the processing element-to-processing element (as opposed to processing element-to-home memory) data transfer mechanism defined by the cache coherence protocol
DONE_INTERVENTION	
DATA_ONLY	
NOT_OWNER	Address conflicts within the system that need resolution

3.4.3.1.7 Packet Scheduler

Packet scheduler handles the scheduling of packet transmission to transport layer. It supports both Round Robin and Weighted Round Robin mechanisms.

3.4.3.2 Packet Handler

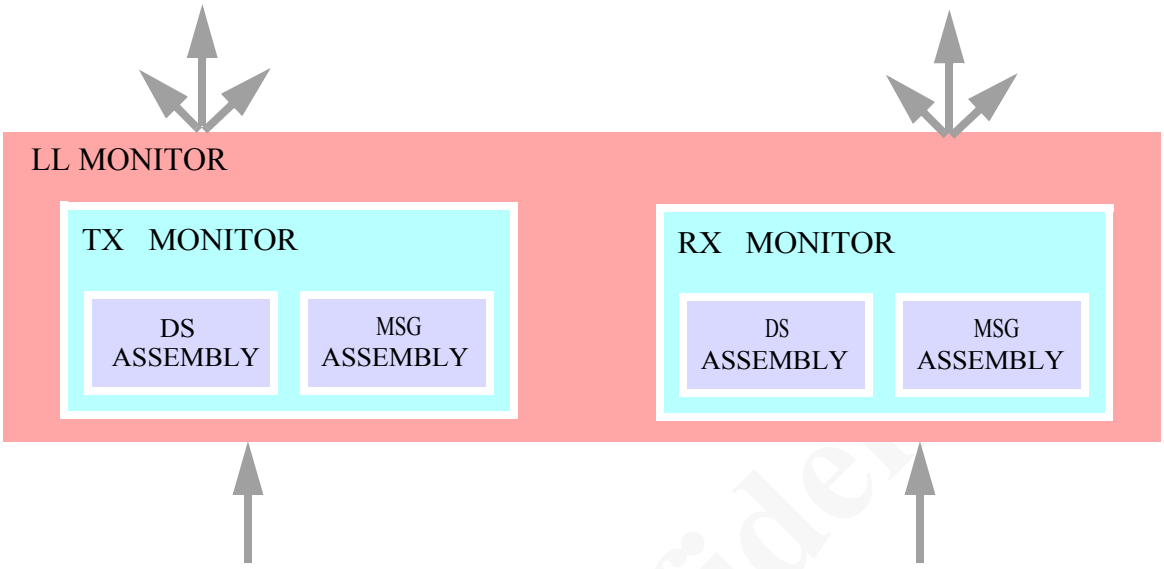
Packet handler processes the different types of packets which are received on the RX side. If there is any need of response then the packet handler module will intimate the required information to response generator module, present in the logical transaction generator to generate the appropriate responses. Implements memory model which is updated with the data received in memory write transactions. For memory read transactions, data from memory is read and given as response payload. It also takes care of updating the register model with the data received in register access transactions.

3.4.4 LL Monitor

Block diagram of logical layer monitor is shown in Figure 4, “Block Diagram of Logical Layer Monitor,” on page 26. It consists of the following sub-modules.

- TX Monitor
- RX Monitor

Figure 4 : Block Diagram of Logical Layer Monitor



3.4.4.1 TX/RX Monitor

The monitor as the name implies, monitors the transactions and reports the protocol violation from the DUT. If any protocol violation is found in the received transaction, corresponding error detect status bits are updated in the register model. In case of known errors i.e. the errors injected by the user to test the behavior of the device connected on the other end, the monitor that detects those violations will report them as warnings and checks for the necessary behavior from DUT. It contains two sub modules.1. DS assembly and 2. Message assembly.

As data streaming and message may consist of more than one segment, monitor assembles all the segments of the same data stream/message and checks if they are transmitted as per the rules defined by the protocol.

On receiving a request packet from the TL through the analysis port, LL monitor performs protocol checking and updates the register model (if required). If the response is expected, the same is predicted and stored in an array as an outstanding request.

For every new multi segment message and data streaming request, a new context is opened to assemble all the segments. For every requests that requires a response, a response timer is started simultaneously. Once the timer times out, (when the response is not received within the time limit) the corresponding request is removed from the outstanding request array and an error is reported.

On receiving a response packet from the TL through the analysis port, LL monitor performs protocol checking and updates the register model (if required).The actual response is com-

pared with the predicted response and reports error on mismatch. Response timers are disabled and the corresponding entry in the outstanding request array is removed

In case of multi segment message response, ‘Response to request’ timer is started for every received response. They are disabled when the next message segment is received or timeout error is reported.

Refer to Table 16, “Logical Layer Checks,” on page 85 for list of checks implemented in logical layer monitor.

3.5 Transport Layer Agent

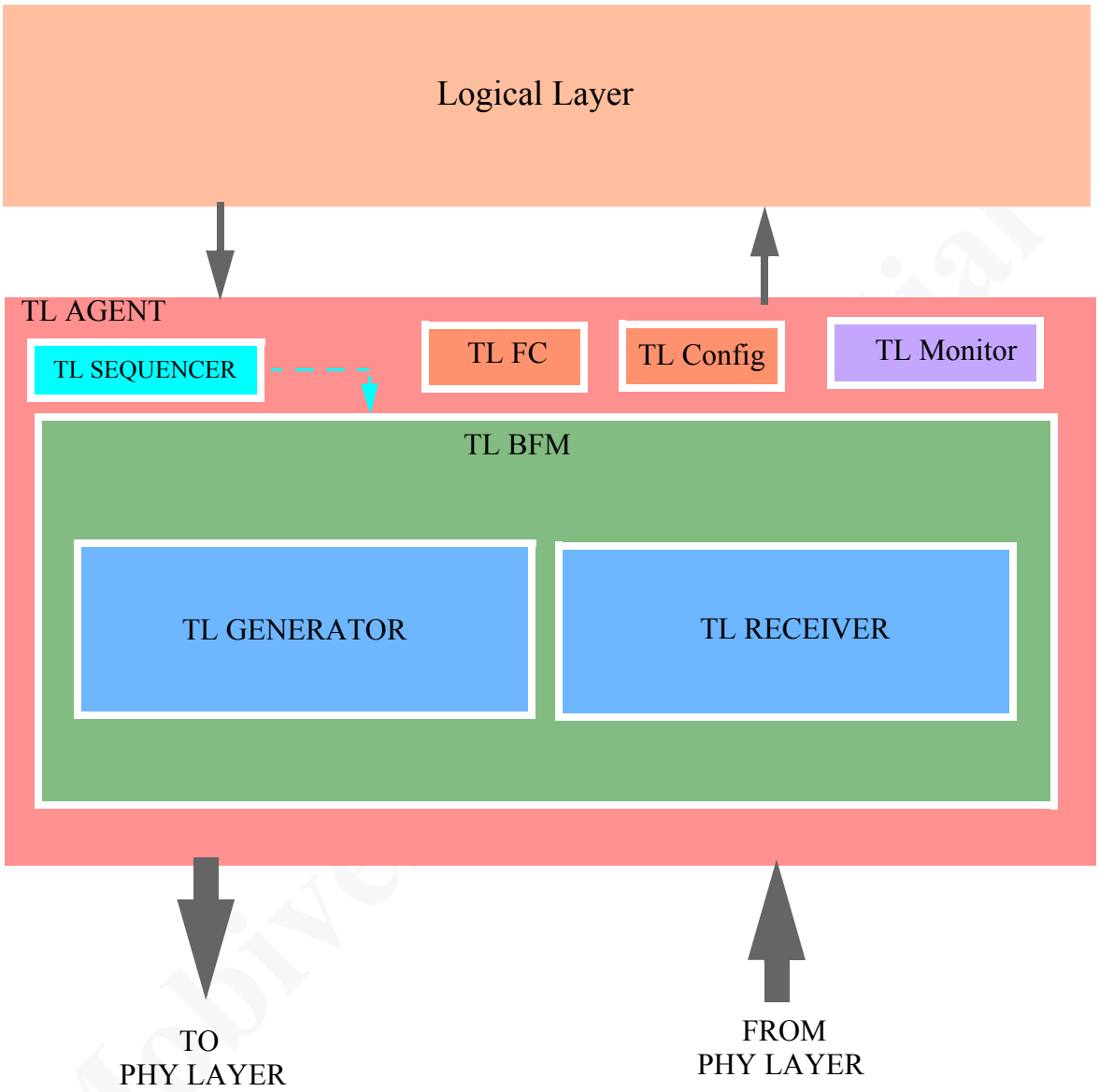
The transport layer agent (srio_tl_agent) implements the functionality of transport layer of the Serial RapidIO standards and performs the function of transmitting and receiving the transport layer data. srio_tl_agent is extended from the UVM base class uvm_agent and its parent component is srio_env.

Transport layer agent instantiates the following modules:

- TL Config
- TL Sequencer
- TL BFM
- TL Monitor

Figure 5, “Block Diagram of Transport Layer Agent,” on page 29 shows the transport layer agent.

Figure 5 : Block Diagram of Transport Layer Agent



3.5.1 TL Configuration

Transport Layer configuration object (srio_tl_config) is inherited from uvm_object, has the configuration variables used by the components of transport layer agent.

Refer to Table 9, “Transport Layer Configuration Parameters,” on page 30 for list of transport layer configuration variables and their description.

Table 9: Transport Layer Configuration Parameters

S.NO	Parameter	Description
1	is_active	UVM_ACTIVE - BFM component will be instantiated and drive the interface signals. UVM_PASSIVE - Only Monitor component will be instantiated.Default - UVM_ACTIVE.
2	has_checks	If TRUE, enables the TL checks in the TL monitor else checks are disabled.
3	has_coverage	If TRUE, functional coverage for transport layer is enabled.
4	en_deviceid_chk	If set to 1, i.e when the Device ID check is enabled, Destination ID of the received packet is expected to match with its own Device ID, failing which will result in uvm_error and the packet will not be forwarded to the upper layer (LL) and hence it will be discarded.
5	usr_sourceid_en	Enable/Disable using Source ID value from user.Default - FALSE.
6	usr_destinationid_en	Enable/Disable using Destination ID value from user.Default - FALSE.
7	usr_sourceid	Source ID value from user.
8	usr_destionationid	Destination ID value from user.
9	lfc_orphan_timer	Orphan timer value for LFC packets in Nano seconds.Default - 10000.

3.5.2 TL Sequencer

The Transport Layer agent sequencer has a sequence driver and a sequence item.The role of the sequencer is to route the sequence item from a sequence to the BFM.The sequence item is of class type srio_trans and the fields are shown in Table 15, “SRIO VIP’s Sequence Item,” on page 78.

3.5.3 TL BFM

Transport Layer BFM is mainly used to generate and receive transport layer fields of a packet as per the configuration.The sub-modules involved in transport layer BFM are listed below.

- TL Generator
- TL Receiver

3.5.3.1 TL Generator

TL generator receives the packet from the logical layer or through transport layer sequencer and adds the following transport layer fields and transmits to the physical layer.If user mode is enabled, then the values provided by the user will be used or the random values generated during the sequence item generation will be used.TL generator also handles the flow control and DS traffic management logic.

- Source ID
- Destination ID
- Transport Type

3.5.3.2 TL Receiver

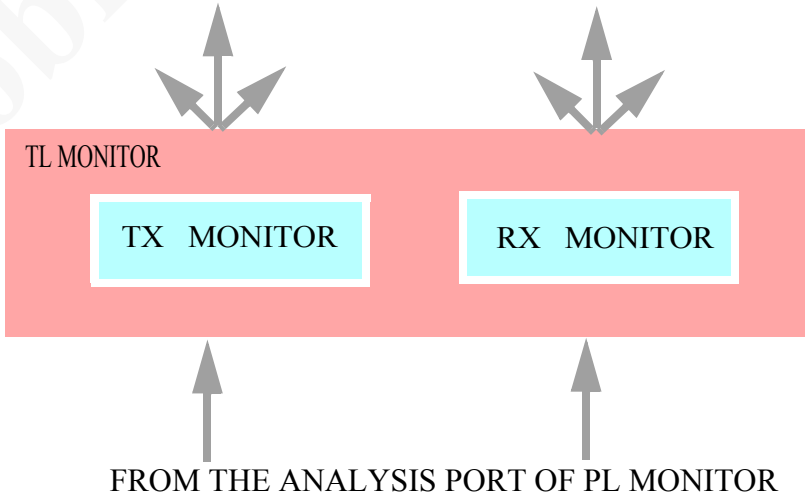
TL receiver module receives the packet from physical layer and collects the relevant transport layer fields such as Source ID, Destination ID and Transport Type (tt) and passes it to the logical layer.

3.5.4 TL Monitor

The transport layer monitor handles the protocol checking related to transport layer functionality. Figure 6, “Block Diagram of Transport Layer Monitor,” on page 31 shows the block diagram of TL monitor. Following components are part of TL monitor.

- Tx Monitor
- Rx Monitor

Figure 6 : Block Diagram of Transport Layer Monitor



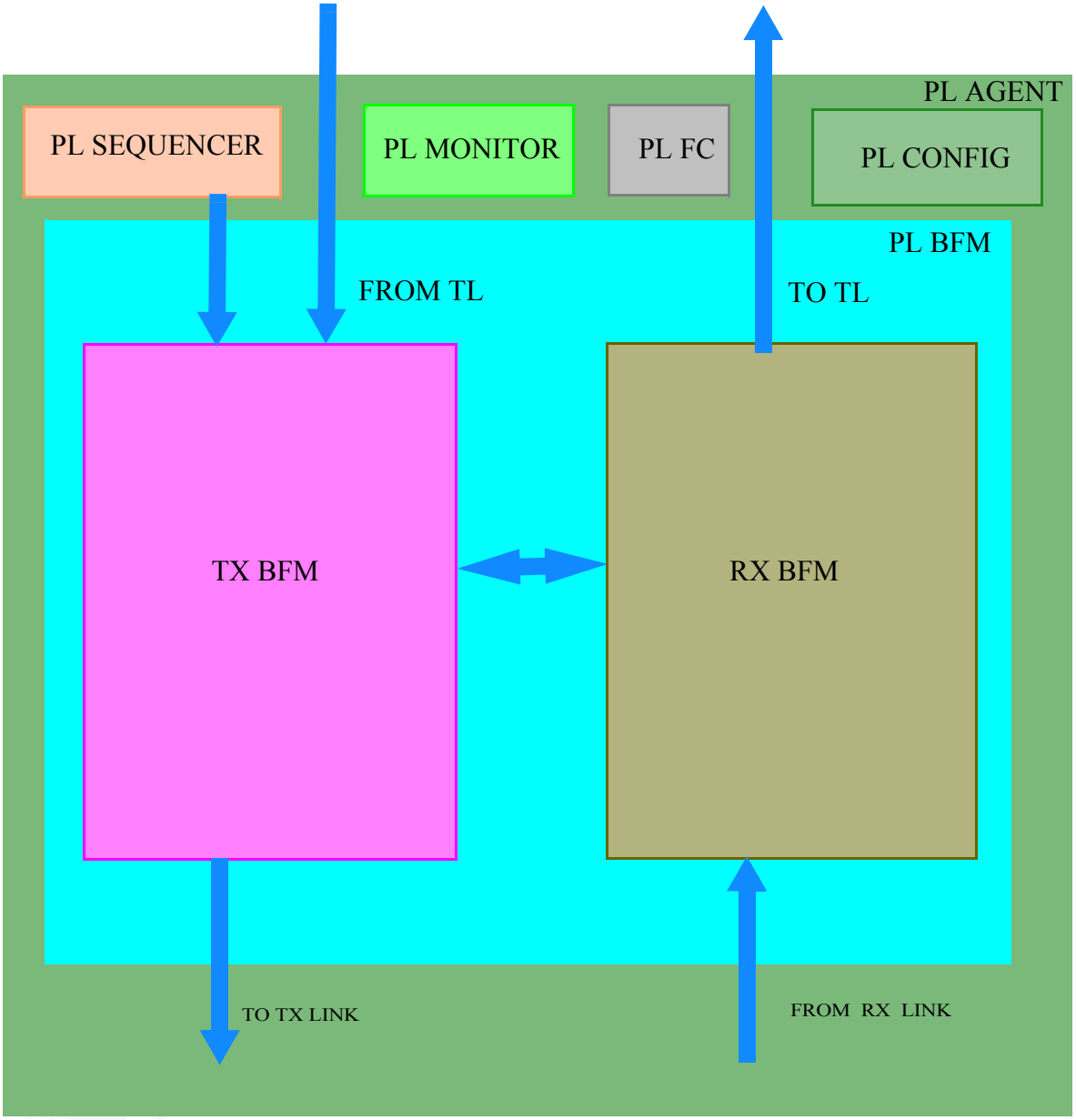
The Transport layer monitor has two subcomponents, TX monitor and RX monitor. These components monitor the transactions and reports the protocol violations from DUT. If any protocol violation is found in the transaction, corresponding error detect status bits are updated in the register model. In case of known errors i.e. the errors injected by the user to test the behavior of the device connected on the other end, the monitor that detects those violations will report them as warnings instead of reporting them as errors. Refer to Table 17, “Transport Layer Checks,” on page 91 for the list of checks implemented in transport layer monitor.

3.6 Physical Layer Agent

The physical layer agent (srio_pl_agent) implements the functionalities of the physical layer defined in the Serial RapidIO standards. The physical layer agent performs the function of transmitting and receiving the physical layer data. srio_pl_agent is extended from the UVM base class uvm_agent and its parent component is srio_env.

The architecture of the physical layer agent is shown in Figure 7, “Block Diagram of Physical Layer Agent,” on page 34.

Figure 7 : Block Diagram of Physical Layer Agent



The contents of the physical layer agent include

- PL Config
- PL Sequencer
- PL Monitor
- PL BFM

3.6.1 PL Configuration

Physical Layer configuration object (srio_pl_config) is inherited from uvm_object, has the configuration variables used by the components of physical layer agent.

Refer to Figure 10, “PL Agent Configuration Parameters,” on page 35 for list of physical layer configuration variables and their description.

Table 10: PL Agent Configuration Parameters

S.No	Parameter	Description
1	has_checks	If TRUE, enables the PL checks in the PL monitor else checks are disabled.
2	has_coverage	If TRUE, functional coverage for physical layer is enabled.
3	comma_cnt_threshold	Comma Count Threshold. Number of comma(Idle_k) symbols needed for synchronization.Default value is 127
4	clk_compensation_seq_rate	Clock compensation sequence rate. Default value is 4096
5	ism_status_cs_sent	Number of status control symbol sent in initialization phase prior to entering normal operational mode.Default value is 15
6	ism_status_cs_rx	Number of error free control symbols received to enter into normal operational mode.Default value is 7
7	sync_break_threshold	Sync Break Threshold.Number of invalid symbols needed to break the sync.
8	lane_misalign_threshold	Lane Misalignment Threshold. Number of invalid A characters to declare lane misalignment.
9	code_group_sent_2_cs	Number of allowed code groups sent between two status control symbols
10	tx_scr_en	Transmit scrambler enable.When true scrambling is enabled.When false, scrambling is disabled
11	vmin_sync_threshold	VMIN Sync Threshold. Threshold for valid number of successive symbols needed in establishing synchronization in initialization phase.
12	valid_sync_threshold	Valid Sync Threshold. Number of valid characters after an invalid character reception threshold

S.No	Parameter	Description
13	bfm_discovery_timer	Discovery timer value used by the BFM. It is executed based on the posedge of sim_clk. Thus, in order to match the DUT's timer value, the model's timer value have to be programmed based on the sim_clk period. For e.g., if DUT's discovery timer value is programmed as 20000ns for simulation purpose, and if the sim_clk period is 2ns, then the discovery timer value in pl agent config has to be set as 10000.
14	bfm_silence_timer	Silence timer value used by the BFM. It also need to be programmed in the same way as described for discovery timer.
15	lp_discovery_timer	Discovery timer value used by the BFM's link partner. It also need to be programmed in the same way as described for discovery timer.
16	lp_silence_timer	Silence timer value used by the BFM's link partner. It also need to be programmed in the same way as described for discovery timer.
17	pkt_retry_support	Packet Retry Support.When true, packet retry support is enabled.When false, packet retry support is disabled.
18	idle_seq_check_en	Idle Sequence Check Enable.When true, check is enabled. When false, check is disabled
19	force_reinit_en	Force Re-initialization Enable.When true, force re-initialization is enabled.
20	ackid_threshold	ACKID Threshold. Threshold for the number of outstanding packets that will return response
21	aet_en	Adaptive Equalization Training enable.When true AET is performed. When false AET is not performed
22	force_1x_mode_en	Force 1x Mode Enable. when true,enables force 1x mode support. Enabled by default.

S.No	Parameter	Description
23	force_laner_en	Force LaneR Mode Enable.When true, enables the support for force laneR. It is valid only if force_1x_mode_en is true. Enabled by default.
24	buffer_space	Buffer Space. Buffer space when flow control mode set to transmit flow control.Default value set is 16
25	buffer_rel_min_val	Buffer Space release time is randomly selected between minimum and maximum values. Time is in terms of clock cycles.
26	buffer_rel_max_val	Buffer Space release time is randomly selected between minimum and maximum values. Time is in terms of clock cycles.
27	flow_control_mode	Flow Control Mode.(SRIO_FC_TRANSMIT -transmit control enable ,SRIO_FC_RECEIVE-receive control flow control enable
28	aet_command_period	Configures the time period between two successive AET commands.(Period.is mentioned in terms of nano seconds)
29	cs_field_ack_timer	CS Field ACK Timer. When AET is enabled time within which the ACK will be received for a training command
30	align_threshold	Align Threshold. Threshold for the number of valid A columns needed for declaring lane alignment
31	default_cs_stype0	Default CS Stype0. Stype0 set to status when generating stype1 control symbol
32	default_cs_stype1	Default CS Stype1. Default stype1 set to NOP when generating stype0 control symbol
33	link_timeout	Port Link Timeout.Port Link timeout value
34	vc_refresh_interval	VC Refresh Interval.
35	vc_status_cs_rate	VC Status CS Rate.
36	pkt_accept_prob	Packet Accepted Probability.Probablility for sending out Packet Accepted control symbol
37	pkt_na_prob	Packet Not Accepted Probability.Probablility for sending out Packet Not Accepted control symbol

S.No	Parameter	Description
38	pkt_retry_prob	Packet Retry Probability.Probablility for sending out Packet Retry control symbol
39	brc3_training_mode	Training Mode. When set, long run training mode is supported, else, short run training mode is supported.Specific to Baud Rate Class3. Default is short-run support.
40	tap_minus_min_value	Tap(_) Minimum Value.
41	tap_minus_max_value	Tap(_) Maximum Value.
42	tap_minus_rst_value	Tap(_) Reset Value.
43	tap_minus_prst_value	Tap(_) Preset Value.
44	tap_plus_min_value	Tap(+) Minimum Value.
45	tap_plus_max_value	Tap(+) Maximum Value.
46	tap_plus_rst_value	Tap(+) Reset Value.
47	tap_plus_prst_value	Tap(+) Preset Value.
48	def_tap	Default Tap
49	tap_rst_value	Tap Reset Value
50	tap_preset_value	Tap Preset Value
51	aet_cmd_kind	AET command kind {CMD_ENABLED,CMD_DISABLED}.
52	aet_cmd_cnt	Total number of aet command sending count.
53	aet_cmd_type	AET command type {TAPPLUS,TAPMINUS,RST,PRST,CMD_RANDOM}.
54	aet_tplus_kind	AET tap plus kind {TP_HOLD,TP_INCR,TP_DECR,TP_RANDOM}.
55	aet_tminus_kind	AET tap minus kind {TM_HOLD,TM_INCR,TM_DECR,TM_RANDOM}.
56	aet_training_period	The time period after which the receiver trained will be asserted when no training command is received
57	vc_ct_mode	VC continuous traffic mode.When set,VC set to CT mode and when not set VC set to RT mode
58	idle2_data_field_len	Length of Idle2 sequence Data field Length
59	max_pkt_size	Maximum packet size

S.No	Parameter	Description
60	pkt_ack_gen_mode	Packet acknowledgement generation kind {PL_IMMEDIATE, PL_RANDOM and PL_DISABLED}.
61	pkt_ack_delay_min	Minimum value of packet acknowledgement transmission delay.
62	pkt_ack_delay_max	Maximum value of packet acknowledgement transmission delay.
63	pl_response_gen_mode	Packet response generation kind {IMMEDIATE,RANDOM and DISABLED}
64	pl_response_delay_min	Minimum value of PL response packet transmission delay.
65	pl_response_delay_max	Maximum value of PL response packet transmission delay.
66	response_en	Response Enable. When true,response driving is enabled and when false driving response is disabled
67	ackid_status_pnack_support	ACKID Status PNACK Support.When true, Param0 field of Packet Not Accepted CS carry AckId status information.When false,Param0 field of Packet Not Accepted CS does not carry AckId status information
68	timestamp_sync_support	Timestamp Sync Support.When true,timestamp sync is supported. When false,timestamp sync is not supported. Specific to specification revision 3.0.
69	timestamp_master_slave_support	Timestamp Master Slave support.When TRUE, timestamp master is supported and when false timestamp slave is supported.
70	seed_ord_seq_rate	Seed ordered sequence rate.The rate at which seed order sequences are transmitted in IDLE3 sequence.Default value is 48
71	status_cntl_ord_seq_rate_min	Minimum code words allowed in between 2 status/control ordered sequence. Default value is 18.
72	status_cntl_ord_seq_rate_max	Maximum code words allowed in between 2 status/control ordered sequence. Default value is 49.

S.No	Parameter	Description
73	asymmetric_support	Asymmetric mode support.When true,asymmerty is supported. When false, asymmetry is not supported.Specific to Baud Rate Class3
74	cs_merge_en	If TRUE, control symbols carries both STYPE0 and STYPE1 functionalities. If FALSE, packed and non-status control symbols are transmitted on an individual basis.
75	cs_embed_en	If TRUE, control symbols are embedded with in packets.If FALSE,embedded CS's are disabled.
76	skew_en	Array of 16 bits, where each bit position indicates corresponding lane number. If an array bit is set, skew is enabled on that particular lane. Default value is 0 on all positions. 0 to 70 in serial and GEN1/2 mode.0 to 469 in serial and GEN3 mode.0 to 7 in parallel and GEN1/2/3 mode.
77	skew_min	Skew range minimum value. Integer type unpacked array of 16 locations.
78	skew_max	Skew range maximum value. Integer type unpacked array of 16 locations.
79	idle_sel	If TRUE, idle2 is enabled else idle1 is enabled.
80	nx_mode_support	Indicates NX mode is supported or not. A value of '1' indicates nx mode is supported.
81	x2_mode_support	Indicates X2 mode is supported or not. A value of '1' indicates 2x mode is supported.
82	brc3_v_cnt_threshold	BRC3 V_counter threshold. Used in code word lock state machine to count the valid code words.
83	brc3_iv_slip	BRC3 IV_Slip value used in code word lock state machine.
84	brc3_ds_cnt_threshold	BRC3 DS_counter threshold. Used in sync state machine for BRC3 devices to count the descrambler seed code words.

S.No	Parameter	Description
85	lock_break_threshold	Number of invalid codewords to be received in order to break the codeword lock in a lane.
86	sync1_state_ui_cnt_threshold	Unit interval used in sync state machine for BRC3, to wait before moving to SYNC_2 state from SYNC_1 state.
87	cw_training_ack_timeout_period	ACK/NAK timeout period for codeword training.
88	cw_training_cmd_deassertion_period	Time period within which command has to return to “hold” incase of codeword training.
89	gen3_training_timer	Time period within which codeword training/re-training or DME training is expected to complete.
90	gen3_keep_alive_assert_timer	Time period after which keep_alive signal has to be asserted in any particular lane which is in TRAINED state.
91	gen3_keep_alive_deassert_timer	Time period after which keep_alive signal has to be deasserted in any particular lane once asserted.
92	bfm_dme_training_c0_preset_value	C0 tap preset value used by the BFM for DME training. Default value is 20.
93	bfm_dme_training_c0_init_value	C0 tap initialize value used by the BFM for DME training. Default value is 0.
94	bfm_dme_training_c0_min_value	C0 tap minimum coefficient value used by the BFM for DME training. Default value is 0.
95	bfm_dme_training_c0_max_value	C0 tap maximum coefficient value used by the BFM for DME training. Default value is 20.
96	bfm_dme_training_cp1_preset_value	CP1 tap preset value used by the BFM for DME training. Default value is 0.
97	bfm_dme_training_cp1_init_value	CP1 tap initialize value used by the BFM for DME training. Default value is 0.
98	bfm_dme_training_cp1_min_value	CP1 tap minimum coefficient value used by the BFM for DME training. Default value is 0.
99	bfm_dme_training_cp1_max_value	CP1 tap maximum coefficient value used by the BFM for DME training. Default value is 15.

S.No	Parameter	Description
100	bfm_dme_training_cn1_preset_value	CN1 tap preset value used by the BFM for DME training. Default value is 0.
101	bfm_dme_training_cn1_init_value	CN1 tap initialize value used by the BFM for DME training. Default value is 0.
102	bfm_dme_training_cn1_min_value	CN1 tap minimum coefficient value used by the BFM for DME training. Default value is 0.
103	bfm_dme_training_cn1_max_value	CN1 tap maximum coefficient value used by the BFM for DME training. Default value is 10.
104	lp_dme_training_c0_preset_value	C0 tap preset value used by the BFM's link partner for DME training. Default value is 20.
105	lp_dme_training_c0_init_value	C0 tap initialize value used by the BFM's link partner for DME training. Default value is 0.
106	lp_dme_training_c0_min_value	C0 tap minimum coefficient value used by the BFM's link partner for DME training. Default value is 0.
107	lp_dme_training_c0_max_value	C0 tap maximum coefficient value used by the BFM's link partner for DME training. Default value is 20.
108	lp_dme_training_cp1_preset_value	CP1 tap preset value used by the BFM's link partner for DME training. Default value is 0.
109	lp_dme_training_cp1_init_value	CP1 tap initialize value used by the BFM's link partner for DME training. Default value is 0.
110	lp_dme_training_cp1_min_value	CP1 tap minimum coefficient value used by the BFM's link partner for DME training. Default value is 0.
111	lp_dme_training_cp1_max_value	CP1 tap maximum coefficient value used by the BFM's link partner for DME training. Default value is 15.
112	lp_dme_training_cn1_preset_value	CN1 tap preset value used by the BFM's link partner for DME training. Default value is 0.
113	lp_dme_training_cn1_init_value	CN1 tap initialize value used by the BFM's link partner for DME training. Default value is 0.

S.No	Parameter	Description
114	lp_dme_training_cn1_min_value	CN1 tap minimum coefficient value used by the BFM's link partner for DME training. Default value is 0.
115	lp_dme_training_cn1_max_value	CN1 tap maximum coefficient value used by the BFM's link partner for DME training. Default value is 10.
116	dme_cmd_kind	DME Training Command kind {DME_CMD_ENABLED,DME_CMD_DISABLED}
117	dme_cmd_cnt	Total number of DME training command sending count.
118	dme_cmd_type	DME Training command type {DME_HOLD,DME_DECR,DME_INCR,DME_INIT,DME_PRST,DME_CMD_RANDOM}
119	dme_tap_type	DME tap type {DME_CMD_COEF0,DME_CMD_COEFPLUS1,DME_CMD_COEFMINUS1}
120	dme_coef0_kin	DME coefficient0 kind {COEF0_INCR,COEF0_DECR,COEF0_INIT,COEF0_PRST,COEF0_RANDOM}
121	dme_coefplus1_kind	DME coefficientplus1 kind {COEFPLUS1_INCR,COEFPLUS1_DECR,COEFPLUS1_INIT,COEFPLUS1_PRST,COEFPLUS1_RANDOM}
122	dme_coefminus1_kind	DME coefficientminus1 kind {COEFMINUS1_INCR,COEFMINUS1_DECR,COEFMINUS1_INIT,COEFMINUS1_PRST,COEFMINUS1_RANDOM}
123	dme_wait_timer_frame_cnt	DME Training wait frame count.
124	cw_cmd_kind	Codeword training command kind { CW_CMD_ENABLED,CW_CMD_DISABLED }
125	cw_cmd_cnt	Codeword training command count.
126	cw_cmd_type	Codeword training command type {CW_HOLD,CW_DECR,CW_INCR,CW_RSVD1,CW_RSVD2,CW_INIT,CW_PRST,CW_SPC_CMD_STAT,CW_CMD_RANDOM }

S.No	Parameter	Description
127	cw_tap_type	Codeword training tap type {CW_CMD_TAP0,CW_CMD_TPLUS1,CW_CMD_TPLUS2,CW_CMD_TPLUS3,CW_CMD_TPLUS4,CW_CMD_TPLUS5,CW_CMD_TPLUS6,CW_CMD_TPLUS7,CW_CMD_TMINUS8,CW_CMD_TMINUS7,CW_CMD_TMINUS6,CW_CMD_TMINUS5,CW_CMD_TMINUS4,CW_CMD_TMINUS3,CW_CMD_TMINUS2,CW_CMD_TMINUS1}
128	cw_tp0_kind	Codeword tap0 kind {TP0_INCR,TP0_DECR,TP0_INIT,TP0_PRST,TP0_RANDOM}
129	cw_tplus1_kind	Codeword tap plus1 kind {TPLUS1_INCR,TPLUS1_DECR,TPLUS1_INIT,TPLUS1_PRST,TPLUS1_RANDOM}
130	cw_tplus2_kind	Codeword tap plus2 kind {TPLUS2_INCR,TPLUS2_DECR,TPLUS2_INIT,TPLUS2_PRST,TPLUS2_RANDOM}
131	cw_tplus3_kind	Codeword tap plus3 kind {TPLUS3_INCR,TPLUS3_DECR,TPLUS3_INIT,TPLUS3_PRST,TPLUS3_RANDOM}
132	cw_tplus4_kind	Codeword tap plus4 kind {TPLUS4_INCR,TPLUS4_DECR,TPLUS4_INIT,TPLUS4_PRST,TPLUS4_RANDOM}
133	cw_tplus5_kind	Codeword tap plus5 kind {TPLUS5_INCR,TPLUS5_DECR,TPLUS5_INIT,TPLUS5_PRST,TPLUS5_RANDOM}
134	cw_tplus6_kind	Codeword tap plus6 kind {TPLUS6_INCR,TPLUS6_DECR,TPLUS6_INIT,TPLUS6_PRST,TPLUS6_RANDOM}
135	cw_tplus7_kind	Codeword tap plus7 kind {TPLUS7_INCR,TPLUS7_DECR,TPLUS7_INIT,TPLUS7_PRST,TPLUS7_RANDOM}

S.No	Parameter	Description
136	cw_tminus8_kind	Codeword tap minus8 kind {TMINUS8_INCR,TMINUS8_DECR,TMINUS8_INIT,TMINUS8_PRST,TMINUS8_RANDOM}
137	cw_tminus7_kind	Codeword tap minus7kind {TMINUS7_INCR,TMINUS7_DECR,TMINUS7_INIT,TMINUS7_PRST,TMINUS7_RANDOM}
138	cw_tminus6_kind	Codeword tap minus6 kind {TMINUS6_INCR,TMINUS6_DECR,TMINUS6_INIT,TMINUS6_PRST,TMINUS6_RANDOM}
139	cw_tminus5_kind	Codeword tap minus5 kind {TMINUS5_INCR,TMINUS5_DECR,TMINUS5_INIT,TMINUS5_PRST,TMINUS5_RANDOM}
140	cw_tminus4_kind	Codeword tap minus4 kind {TMINUS4_INCR,TMINUS4_DECR,TMINUS4_INIT,TMINUS4_PRST,TMINUS4_RANDOM}
141	cw_tminus3_kind	Codeword tap minus3 kind {TMINUS3_INCR,TMINUS3_DECR,TMINUS3_INIT,TMINUS3_PRST,TMINUS3_RANDOM}
142	cw_tminus2_kind	Codeword tap minus2 kind {TMINUS2_INCR,TMINUS2_DECR,TMINUS2_INIT,TMINUS2_PRST,TMINUS2_RANDOM}
143	cw_tminus1_kind	Codeword tap minus1 kind {TMINUS1_INCR,TMINUS1_DECR,TMINUS1_INIT,TMINUS1_PRST,TMINUS1_RANDOM}
144	tap0_min_value	Codeword training Tap0 Minimum Value
145	tap0_max_value	Codeword training Tap0 Maximum Value
146	tap0_init_value	Codeword training Tap0 Initialization Value
147	tap0_prst_value	Codeword training Tap0 Preset Value
148	tap0_impl_en	Codeword training Tap0 implementation enable

S.No	Parameter	Description
149	tplus1_min_value	Codeword training Tapplus1 Minimum Value
150	tplus1_max_value	Codeword training Tapplus1 Maximum Value
151	tplus1_init_value	Codeword training Tapplus1 Initialization Value
152	tplus1_prst_value	Codeword training Tapplus1 Preset Value
153	tplus1_impl_en	Codeword training Tapplus1 implementation
154	tplus2_min_value	Codeword training Tapplus2 Minimum Value
155	tplus2_max_value	Codeword training Tapplus2 Maximum Value
156	tplus2_init_value	Codeword training Tapplus2 Initialization Value
157	tplus2_prst_value	Codeword training Tapplus2 Preset Value
158	tplus2_impl_en	Codeword training Tapplus2 implementation
159	tplus3_min_value	Codeword training Tapplus3 Minimum Value
160	tplus3_max_value	Codeword training Tapplus3 Maximum Value
161	tplus3_init_value	Codeword training Tapplus3 Initialization Value
162	tplus3_prst_value	Codeword training Tapplus3 Preset Value
163	tplus3_impl_en	Codeword training Tapplus3 implementation
164	tplus4_min_value	Codeword training Tapplus4 Minimum Value
165	tplus4_max_value	Codeword training Tapplus4 Maximum Value
166	tplus4_init_value	Codeword training Tapplus4 Initialization Value
167	tplus4_prst_value	Codeword training Tapplus4 Preset Value
168	tplus4_impl_en	Codeword training Tapplus4 implementation
169	tplus5_min_value	Codeword training Tapplus5 Minimum Value

S.No	Parameter	Description
170	tplus5_max_value	Codeword training Tapplus5 Maximum Value
171	tplus5_init_value	Codeword training Tapplus5 Initialization Value
172	tplus5_prst_value	Codeword training Tapplus5 Preset Value
173	tplus5_impl_en	Codeword training Tapplus5 implementation
174	tplus6_min_value	Codeword training Tapplus6 Minimum Value
175	tplus6_max_value	Codeword training Tapplus6 Maximum Value
176	tplus6_init_value	Codeword training Tapplus6 Initialization Value
177	tplus6_prst_value	Codeword training Tapplus6 Preset Value
178	tplus6_impl_en	Codeword training Tapplus6 implementation
179	tplus7_min_value	Codeword training Tapplus7 Minimum Value
180	tplus7_max_value	Codeword training Tapplus7 Maximum Value
181	tplus7_init_value	Codeword training Tapplus7 Initialization Value
182	tplus7_prst_value	Codeword training Tapplus7 Preset Value
183	tplus7_impl_en	Codeword training Tapplus7 implementation
184	tminus8_min_value	Codeword training Tapminus8 Minimum Value
185	tminus8_max_value	Codeword training Tapminus8 Maximum Value
186	tminus8_init_value	Codeword training Tapminus8 Initialization Value
187	tminus8_prst_value	Codeword training Tapminus8 Preset Value
188	tminus8_impl_en	Codeword training Tapminus8 implementation enable
189	tminus7_min_value	Codeword training Tapminus7 Minimum Value
190	tminus7_max_value	Codeword training Tapminus7 Maximum Value

S.No	Parameter	Description
191	tminus7_init_value	Codeword training Tapminus7 Initialization Value
192	tminus7_prst_value	Codeword training Tapminus7 Preset Value
193	tminus7_impl_en	Codeword training Tapminus7 implementation enable
194	tminus6_min_value	Codeword training Tapminus6 Minimum Value
195	tminus6_max_value	Codeword training Tapminus6 Maximum Value
196	tminus6_init_value	Codeword training Tapminus6 Initialization Value
197	tminus6_prst_value	Codeword training Tapminus6 Preset Value
198	tminus6_impl_en	Codeword training Tapminus6 implementation enable
199	tminus5_min_value	Codeword training Tapminus5 Minimum Value
200	tminus5_max_value	Codeword training Tapminus5 Maximum Value
201	tminus5_init_value	Codeword training Tapminus5 Initialization Value
202	tminus5_prst_value	Codeword training Tapminus5 Preset Value
203	tminus5_impl_en	Codeword training Tapminus5 implementation enable
204	tminus4_min_value	Codeword training Tapminus4 Minimum Value
205	tminus4_max_value	Codeword training Tapminus4 Maximum Value
206	tminus4_init_value	Codeword training Tapminus4 Initialization Value
207	tminus4_prst_value	Codeword training Tapminus4 Preset Value
208	tminus4_impl_en	Codeword training Tapminus4 implementation enable
209	tminus3_min_value	Codeword training Tapminus3 Minimum Value
210	tminus3_max_value	Codeword training Tapminus3 Maximum Value
211	tminus3_init_value	Codeword training Tapminus3 Initialization Value

S.No	Parameter	Description
212	tminus3_prst_value	Codeword training Tapminus3 Preset Value
213	tminus3_impl_en	Codeword training Tapminus3 implementation enable
214	tminus2_min_value	Codeword training Tapminus2 Minimum Value
215	tminus2_max_value	Codeword training Tapminus2 Maximum Value
216	tminus2_init_value	Codeword training Tapminus2 Initialization Value
217	tminus2_prst_value	Codeword training Tapminus2 Preset Value
218	tminus2_impl_en	Codeword training Tapminus2 implementation enable
219	tminus1_min_value	Codeword training Tapminus1 Minimum Value
220	tminus1_max_value	Codeword training Tapminus1 Maximum Value
221	tminus1_init_value	Codeword training Tapminus1 Initialization Value
222	tminus1_prst_value	Codeword training Tapminus1 Preset Value
223	tminus1_impl_en	Codeword training Tapminus1 implementation enable
224	tminus1_min_value	Codeword training Tapminus1 Minimum Value
225	tminus1_max_value	Codeword training Tapminus1 Maximum Value
226	tminus1_init_value	Codeword training Tapminus1 Initialization Value
227	tminus1_prst_value	Codeword training Tapminus1 Preset Value
228	tminus1_impl_en	Codeword training Tapminus1 implementation enable
229	k_cnt_for_idle_detection	Number of K characters to match in-order to detect the receiving IDLE sequence. Default value is 4.

S.No	Parameter	Description
230	m_cnt_for_idle_detection	Number of M characters to match in-order to detect the receiving IDLE sequence. Default value is 5.
231	gen3_max_pkt_size	Maximum packet size to be checked for GEN3 devices.
232	timestamp_auto_update_en	Enable for automatically sending timestamp sequence.
233	timestamp_auto_update_timer	Indicates the time in within which the timestamp sequence has to be sent by the master automatically. This field is valid only if both timestamp_sync_support and timestamp_master_slave_support are 1.
234	asym_1x_mode_en	Asymmetric 1x mode enable.
235	asym_2x_mode_en	Asymmetric 2x mode enable.
236	asym_nx_mode_en	Asymmetric Nx mode enable.
237	xmt_width_timer	Timeperiod within which transmit width command is expected to complete. Timeperiod is counted in terms of sim_clk period. Default value is 10000.
238	rcv_width_timer	Timeperiod within which receive width command os expected to complete. Timeperiod is counted in terms of sim_clk period. Default value is 10000.
239	xmt_my_cmd_timer	Timeperiod within which “My transmit width change” command has to be acknowledged. Timeperiod is counted in terms of sim_clk period. Default value is 10000.
240	xmt_lp_cmd_timer	Timeperiod within which “Link-partner transmit width change” command has to be acknowledged. Timeperiod is counted in terms of sim_clk period. Default value is 10000.

3.6.2 PL User Input Parameters

Physical Layer user input parameters are set of control parameters that needs to be configured by the user during run time to exercise any specific functionalities listed in Table 11, “PL User Input Parameters,” on page 51

These parameters are declared in srio_pl_common_component_trans.sv in srio-vip/pl directory. Though many status variables are declared in it, only the control variables need to be configured by the user are listed in the below table.

Table 11: PL User Input Parameters

S.No	Parameter	Description
1	force_1x_mode	Need to set it to force the BFM to operate in 1x mode.
2	force_laneR	Need to set to force the BFM to operate in 1x mode laneR.
3	force_reinit	Need to set to BFM re-initialization.
4	link_req_rst_cmd_cnt	Number of link-request with reset-device command received by the BFM. If the count is 4, user need to execute the logic required for reset-device.
5	link_req_rst_port_cmd_cnt	Number of link-request with reset-port command received by the BFM. If the count is 4, user need to execute the logic required for reset-device.
6	change_my_xmt_width	Need to configure it appropriately to test the asymmetric operation. This field is equivalent of “Change my transmit width” field in PnPMCSR.
7	change_lp_xmt_width	Need to configure it appropriately to test the asymmetric operation. This field is equivalent of “Change link partner transmit width” field in PnPMCSR.
8	timestamp_support	Indicates the timestamp support.
9	timestamp_master	If set, indicates, the BFM acts as timestamp master. This field is valid only if timestamp_support is set. If timestamp_support is set, and timestamp_master is not set, then the BFM acts as timestamp slave.
10	send_zero_timestamp	If this field is set along with timestamp_support and timestamp_master, then the BFM will transmit zero timestamp sequence.

Table 11: PL User Input Parameters

S.No	Parameter	Description
11	send_timestamp	If this field is set along with timestamp_support and timestamp_master, then the BFM will transmit timestamp sequence loaded with TSG value + timestamp offset.
12	send_loop_request	If this field is set along with timestamp_support and timestamp_master, then the BFM will transmit loop-timing request control symbol.

3.6.3 PL Sequencer

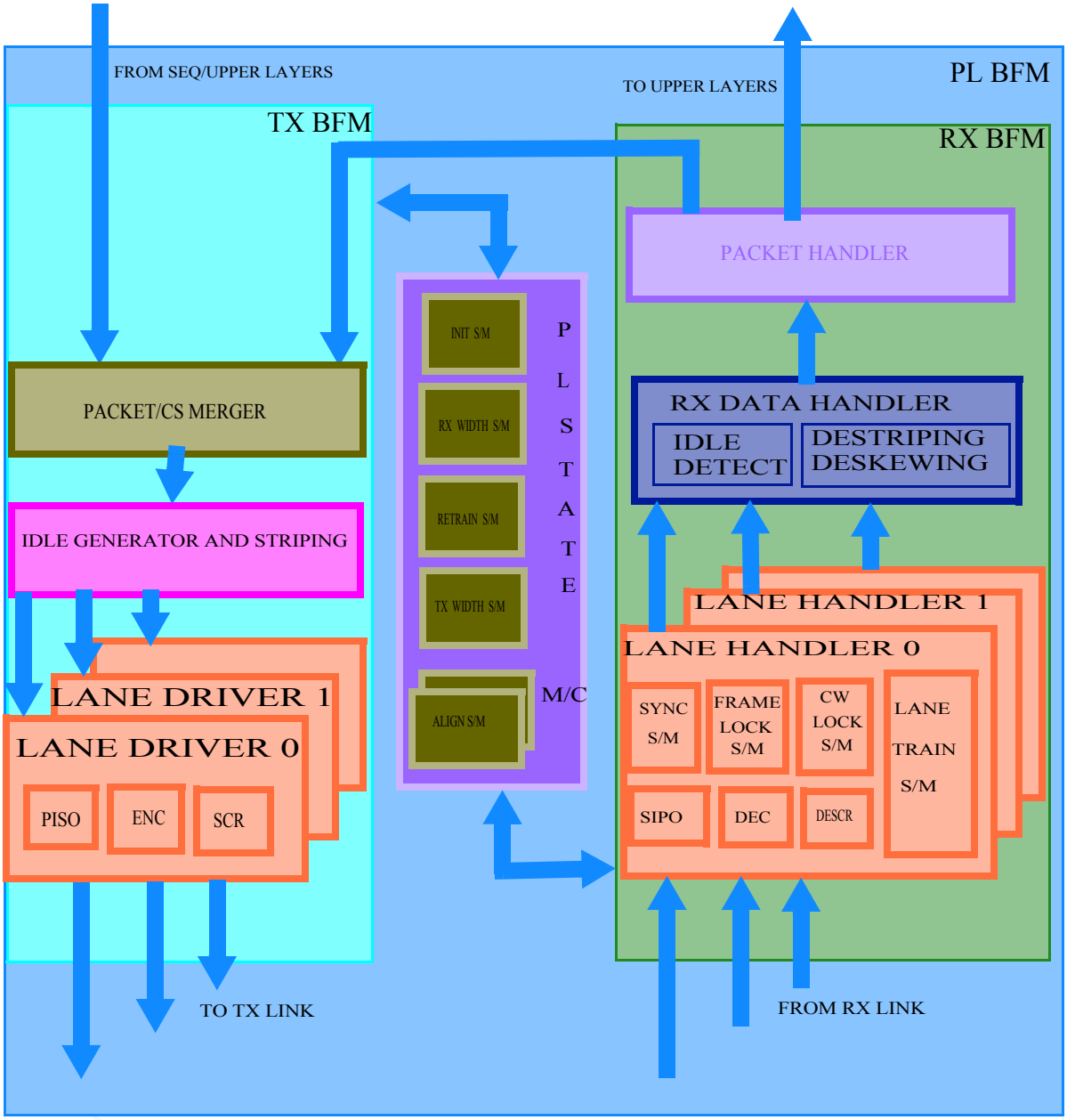
The physical layer sequencer (srio_pl_sequencer) has a sequence driver and a sequence item.The role of the sequencer is to route the sequence item from a sequence to the bfm.The sequence item is of class type srio_trans and the fields are defined in, Table 15, “SRIO VIP’s Sequence Item,” on page 78:

3.6.4 PL BFM

The physical layer BFM is responsible for converting the data inside the sequence item into pin level transactions.The sequence item of the driver consists of all the fields defined for the serial physical layer, transport layer and the logical layer.The driver will run sequence items of type srio_trans sequence defined in the phy layer sequence and sequence items received from the upper layer agents.It is further divided into Transmit Path and the Receive Path BFM.

Refer to Figure 8, “Block Diagram of Physical Layer BFM,” on page 53 for the block diagram of physical layer BFM.

Figure 8 : Block Diagram of Physical Layer BFM



3.6.4.1 Transmit Path

The transmit path is responsible for converting the sequence item defined in the physical layer sequence or the sequence items received from the upper layer agents to pin level transactions.

3.6.4.1.1 Packet/Control Symbol Generator

In transmit path, sequence items can be received from the physical layer sequence or from the upper layer agents.Sequence items received from the upper layer agents are stored in a TLM FIFO. The transmit driver arbitrates in a round robin method between the sequences defined in the phy layer sequencer and the TLM fifo and converts into SRIO phy packet type or control symbol type or state machine type and are pushed into packet queue based on their priority and control symbol queue respectively for further processing.In addition to the above queues, an internally generated control symbol queue is defined for generating control symbols based on the control symbols/packets received from the receive path.If a state machine type sequence is received, respective signals are triggered for transition from one state to another state.Once the packet is popped from the packet queue a copy of it is stored in the outstanding ackid queue when the transmitter is configured in receiver controlled flow control mode.The re-ordering logic in this block will decide which packet has to be sent on the link when a packet retry control symbol is received.The packet and control symbol handler on the receive path will handle the reception of control symbols and communicate with this block to initiate the control symbols generation as per the Serial RapidIO specification.For example, when a stype0 command is received on a control symbol, it passes on the appropriate information by updating the required field in the common transaction that is being used inside the agent. Based on the information updated, Packet/Control symbol generator will remove a particular packet if packet accepted CS is received or it will re-order and retry the packet if packet retry control symbol is received.

3.6.4.1.2 Packet/Control Symbol Merger

Once the packet queue or the control symbol queue or the internally generated control symbol queue is not empty, merger logic pops the packet from the packet queue and decides whether the control symbols can be embedded into the packet or not by traversing in the respective control symbol queues and if it can be embedded, the control symbols are popped and merged with packets and if it cannot be merged, separate control symbol transaction is formed by popping the control symbol and is converted into list of bytes and pushed in to the separate CS processing queue for further processing.Merged packets with control symbols are converted in to list of bytes and pushed in to packet processing queue.

3.6.4.1.3 IDLE Generator and Striping

The IDLE generator generates the idle1 sequence when the agent is configured in Idle1 mode and idle2 sequence when the agent is configured in Idle2 mode and idle3 when configured in idle3 mode. Clock compensation sequence is automatically generated once in every 4096 idle characters transmitted if idle1/idle2 mode is enabled.Based on the number of lanes configured, the generated idle pattern is stored in the respective lane buffers for further processing.When in link initialized state, if packet processing queue or control symbol processing queue is not empty, the idle sequence is terminated as per the RapidIO specification and based on simple arbitration, either packet processing queue or control symbol processing queue is popped and converts each byte to a character class, stripped and pushed in to respective lane buffers for further processing.The above logic is implemented using a state machine.

By default,idle1/idle2/idle3 sequences are generated when neither packet nor control symbols are available for transmission.Character object is formed for each byte and are fed

into respective lane buffers for transmission on the link.When the port is initialized and packet processing queue or the control symbol processing queue is not empty, the state machine moves to either packet state or control symbol state and forms the character class and are striped and fed into respective lane buffers for transmission.

3.6.4.1.4 Lane Driver

The function of the lane driver class is to get the character object from the idle generator/ striping module and scrambles, encodes, serializes and transmitted on the link using transmit clock. There is a separate lane driver block for each of the active lanes configured in the global environment config.

The lane driver class gets each character object and then scrambled based on the control bit. Based on the control bit set, the character may be or may not be scrambled and is passed on to the encoder task for encoding.

Encoder task encodes the characters based on the control bit in the character class as per 8B/10B encoding procedure and the PISO task converts the 10 bit code group into bit streams and transmits on the link using transmit clock.By default skew is not inserted. If skew is enabled, skew is inserted based on the granularity set and is automatically taken care in the lane driver module. The same process happens for Gen3 transmission also with 64/67B encoding and serializing of 67 bits.

3.6.4.2 Receive Path

The receive path consists of the following blocks:

- Lane Handler
- Deskew and Destriping
- Idle Detector
- Packet/Control Symbol/Idle Handler

3.6.4.2.1 Lane Handler

There is a separate lane handler block for each of the active lanes that the model is supporting.The serial data from the lanes are collected by the respective lane handler module and is initially processed to the SIPO method inside the lane handler module. The SIPO method will lock a particular 10-bit serial data when it detects a comma character pattern in it and updates the 10-bit code group field inside a transaction which is used by the lane handler class methods. This transaction is then processed by the decoder method which will decode the 10-bit code group into an 8-bit character along with it, the control information field of the decoded character is also updated. The control information field will indicate whether the decoded character is a special character or a normal data character based on which the scrambler method will either descramble it or just shift the LFSR without descrambling it. In case of a skip control character, the descrambler will neither shift its LFSR nor will descramble it. The different lane specific state machines will then process the descrambled data and when all the methods have processed it, the copy of this trans-

action is framed and the other modules of the Rx BFM are indicated to process it.The SIPO method has to adjust the 10 bit locking if the sync state machine could not achieve the synchronization properly. Similarly, for baud rate class 3, each of the methods will operate on 64-bit data. The adaptive equalization training commands / acknowledgements for BRC2 and BRC3 are received by this block and are updated in the common component transaction, so that the respective lane driver instance will act accordingly. When no command is sent on the tx side until the aet_training_period timer expires, then the receiver trained bit will be set. If long run training is supported in case of BRC3, the frame is collected in the same serial manner and frame lock will be set when the frame marker (32'hFFFF_0000) is detected. The frame is then decoded as per DME protocol, and the long-run training commands are serviced, if it is supported by the device. Once the training is completed, the lane_driver will send specified number of DME frames before starting the codeword transmission. On the other hand, if long-run training is not supported by the link-partner, this block will parallely look for lane_sync and frame_lock, and proceeds with short-run training once lane_sync is achieved.

Parallel mode is also supported by the SRIO-VIP. It is enabled by configuring the srio_interface_mode as SRIO_PARALLEL in env config instance. If parallel mode is enabled, then proper 10-bit/67-bit parallel data needs to be provided to the VIP. It will not perform any bit-shifting to match the comma character / codeword to achieve lane_sync/ codeword lock. The block will directly try to decode and descramble the received parallel data on each parallel clock period.

Callback for 10-bit/67-bit received codegroup is provided by this block. Also, for IDLE2, callback for received CS field is provided by this block to help the user to control / corrupt the AET process.

3.6.4.2.2 Rx Data Handler

Part of the PCS layer functionalities are performed in lane handler class and the remaining functions such as deskew across lanes, destriping, removal of link level CRC-32 and padding, packet and control symbol decoding are performed in this class.

3.6.4.2.2.1 Deskew & Destripping

The deskew and destriping logic will catch the event triggered by all the lane handler class through one of its method. Parallel threads are used to detect the events from all the available lane handler classes. The deskew logic uses two separate queue structures, one for checking 2x alignement and the other for Nx alignment. Each queue structure will contain per lane queues to deskew the received data. 2x alignment is performed on data from lane 0 and lane 1 whereas Nx alignment is performed parallely on data from lane 0 through lane N. The deskew logic will push the A character (for 1.3 and 2.x) / STATUS-CONTROL codeword (for GEN3) into the first position of that particular lane's queue and waits for 7 characters before which if all the lane queue within its queue structure gets the data, the queue will be popped and a transaction is formed. This transaction will contain the character information from all the active lanes and it'll be used by the PL state machines class to decide on the model's current and next state. On the other hand, if any of the queues within a particular alignment logic, doesn't contain any data, then all the queues within that particular queue structure are cleared. By this method, the data skew is removed.

Once the port initialization is completed, the de-striping logic will de-stripe the data, if required, from the appropriate lane queues and forms the srio transaction which will be pushed in to the queue used in the common class. While de-striping, the number of error free status control symbol received is counted. Similarly, from the Tx path, the number of status control symbol transmitted are updated in a common component transaction instance. When the transmitted and received status control symbol count are matched to set the link-initialized variable, it is done so by set_link_initialization method. It is also taken care to reset the link-initialized variable whenever port_initialized variable goes low. After link-initialized variable is set, the Tx path will start transmitting packets, if it is scheduled.

3.6.4.2.2.2 Idle Detector

The IDLE detector logic will use the event captured by the deskew & destriping logic and will detect the IDLE sequence on which the model has to operate upon. The IDLE sequence detection will happen by counting the number of K characters and number of M characters based on the configuration, and once configured count is reached, if K count is higher than M count, then IDLE1 sequence is detected, or else IDLE2 is detected. The detected IDLE sequence information is passed to the TX BFM and TX BFM will switch to the detected IDLE sequence if required. IDLE detection mechanism for BRC3 is not required as it will operate in IDLE3 only.

3.6.4.2.3 Packet/CS Symbol/Idle Handler

Packet/CS Symbol and Idle handler waits for the event triggered from the Rx Data handler.Once the event is triggered, collects the packet or control symbols and information is exchanged to the Tx packet/control symbol generator using a common transcation.If packet is detected, it is processed and pushed to the upper layers for further processing.If a control symbol is detected, the common transaction is updated with the control symbol information received and the Tx Packet/Control symbol generator process the transaction and decides the respective action as described in the Tx Packet/Control symbol Generator block section.

3.6.4.2.4 PL State Machines

This block implements all the physical layer related state machines namely align, Initialization. It also implements other BRC3 specific state machines such as link retraining, Tx and Rx width state machines. The state machine variables are again updated in the common transaction and is used by all the blocks present inside the particular monitor component.

3.6.5 PL Monitor

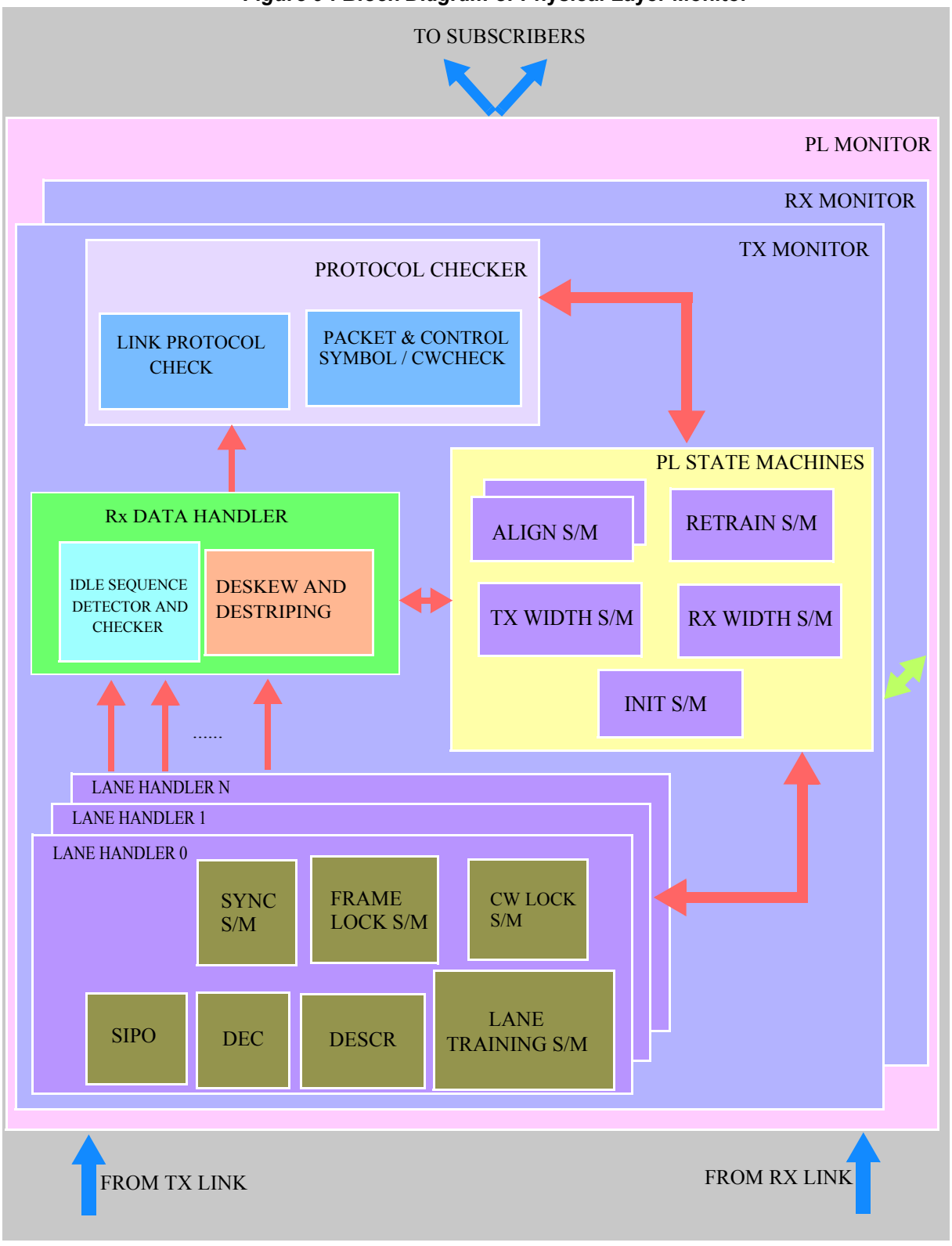
The Physical layer monitor performs the protocol checks on the physical layer packet fields, control symbols and the IDLE sequence which are transmitted and received by the PL agent. It also performs the overall link maintenance protocol checks. The transmit path and receive path of the BFM are appropriately handled by the Tx monitor and Rx monitor instances respectively, which resides inside the PL monitor component. The PL monitor component will contain a sep-

arate handle of a transaction for Tx monitor and the Rx monitor, where all the respective state machine variables, events and other common variables will be declared. The appropriate handle of this transaction will be used by all the components inside the Tx monitor and the Rx monitor. Any of the Tx and Rx path can be configured to report only warnings instead of errors so that any intentional error insertion from the test sequences are not reported as errors, but appropriate protocol checking is performed for such cases. Based on the transactions received by the monitor, register model will be updated appropriately.

The monitor interface could either be configured as DUT or BFM based on which interface it is monitoring. If the monitor interface is configured as DUT, it means the monitor is monitoring the DUT transmit data, and hence all the abnormal functionalities are reported as errors. Since, the received data is same for this monitor and the BFM, this monitor instance will always read the configuration variables for any data processing. The timers used by this monitor will be same as that used by the BFM instance. On the other hand, if the monitor interface is configured as BFM, it means, this monitor will monitor the BFM transmit data (or DUT receive data), and hence all illegal functionalities are reported as warnings. Since, this monitor receives the same data that is received by the DUT, it will poll the register model for all the configurations. Separate timer variables are declared in the configuration instance for BFM monitors, so that these monitors will match the BFM's link partner functionalities and state changes. All required controls need to be properly written in the appropriate registers created in the register model for this monitor instance to operate in the same state as the DUT.

The serial data is converted to a parallel 10bit or 67 bit data based on the Baud rate class in which the model is operating. This parallel data is decoded into an 8-bit character or a 64-bit codeword, which is again based on the baud rate class. It is then descrambled and handled by the lane specific state machines inside the lane handler class and is passed on as a transaction to the Deskew & Destriping block. The IDLE sequence detector & checker logic detects the IDLE sequence and performs all the checks related to it. The deskew and destriping block, aligns the data received on all the active lanes and destripes it based on the initialized mode. The data from the deskew logic is formed as a transaction which contains all the lane's information and passed on to the state machine block and inturn used by the Protocol checker block based on the state of the model. The lane handler, deskew & destriping block and the IDLE sequence detector and checker together with the state machine block forms the PCS / PMA part of the RapidIO physical layer. The check is then performed on the packet or control symbol and is then broadcasts through the analysis port of the monitor. The list of PL checks can be found in the table [Table 18, “Physical Layer Checks,” on page 91.]. The detailed functionality of each of the monitor's sub-component is described below

Figure 9 : Block Diagram of **Physical Layer Monitor**



3.6.5.1 Lane Handler

There is a separate lane handler block for each of the active lanes that the model is supporting. The serial data from the lanes are collected by the respective lane handler module and is initially processed to the SIPO method inside the lane handler module. The SIPO method will lock a particular 10-bit serial data when it detects a comma character pattern in it and updates the 10-bit codegroup field inside a transaction which is used by the lane handler class methods. This transaction is then processed by the decoder method which will decode the 10-bit codegroup into an 8-bit character along with it, the control information field of the decoded character is also updated. The control information field will indicate whether the decoded character is a special character or a normal data character based on which the scrambler method will either descramble it or just shift the LFSR without descrambling it. In case of a skip control character, the descrambler will neither shift its LFSR nor will descramble it. The different lane specific state machines will then process the descrambled data and when all the methods have processed it, the copy of this transaction is framed and the other components of the monitor are indicated to process it by triggering an event.The SIPO method has to adjust the 10 bit locking if the sync state machine could not achieve the synchronization properly. Similarly, for baud rate class 3, 67 bit is locked and each of the methods will operate on 64-bit data. The adaptive equalization training related checks for BRC2 and BRC3 are performed in this block. If long run training is supported in case of BRC3, the frame is collected in the same serial manner and frame lock will be set when the frame marker (32'hFFFF_0000) is detected. The frame is then decoded as per DME protocol, and the long-run training commands are serviced, if it is supported by the device. Once the training is completed, the monitor will wait for specified number of DME frames before starting the codeword transmission. On the other hand, if long-run training is not supported by the link-partner, this block will parallely look for lane_sync and frame_lock, and proceeds with short-run training once lane_sync is achieved.

If parallel mode is enabled, then proper 10-bit/67-bit parallel data will be transmitted by the VIP and the same is expected from the link partner. It will not perform any bit-shifting to match the comma character / codeword to achieve lane_sync/codeword lock. The block will directly try to decode and descramble the received parallel data on each parallel clock period.

Apart from all the above logic implementations, the monitor instance of this block will perform all the checks related to GEN2.x / GEN3 equalization training, lane specific IDLE3 ordered sequence checks and clock-compensation frequency checks for all modes. All the lane specific status register fields are updated by this block based on the configured port number. If any lane specific error is detected by this block, it updates the respective error management related register fields.

3.6.5.2 Rx Data Handler

Part of the PCS layer functionalities are performed in lane handler module and the remaining functions such as deskew across lanes, destriping, removal of link level CRC-32 and padding, packet and control symbol decoding are performed by this component.

The monitor specific instance of this block will update the error detect registers and other error rate management related register fields if any specified error is detected.

3.6.5.3 Deskew & Destriping

The deskew and destriping logic will catch the event triggered by all the lane handler class through one of its method. Parallel threads are used to detect the events from all the available lane handler classes. The deskew logic uses two separate queue structures, one for checking 2x alignement and the other for Nx alignment. Each queue structure will contain per lane queues to deskew the received data. 2x alignment is performed on data from lane 0 and lane 1 whereas Nx alignment is performed parallely on data from lane 0 through lane N. The deskew logic will push the A character into the first position of that particular lane's queue and waits for 7 characters before which if all the lane queue within its queue structure gets the data, the queue will be popped and a transaction is formed. This transaction will contain the character information from all the active lanes and it will be used by the PL state machines class to decide on the model's current and next state. On the other hand, if any of the queues within a particular alignment logic, doesn't contain any data, then all the queues within that particular queue structure are cleared. By this method, the data skew is removed.

Once the port initialization is completed, the de-striping logic will de-stripe the data, if required, from the appropriate lane queues and forms the srio transaction which will be pushed in to the queue used in the common class. While de-striping, the number of error free status control symbol received is counted. Similarly, the number of status control symbol received by the other monitor is also updated in monitor specific common transaction instance. When the counts from both the monitors are matched to set the link-initialized variable, it is done so by set_link_initialization method. It is also taken care to reset the link-initialized variable whenever port_initialized variable goes low.

Apart from all IDLE sequence checks, this model also performs some important checks like any invalid or illegal characters in between packets / control symbols, maximum packet size violation, delimiters check for packets etc. This block will perform the linkcrc32 check on the received packet, and will remove it along with pad bytes if any, and then pushes it into a queue, which is used by the protocol checker block. Timestamp control symbol sequence check is also performed by this block.

3.6.5.4 Idle Detector & Checker

The IDLE detector logic will use the event captured by the deskew & destriping logic and will detect the IDLE sequence on which the model has to operate upon. The TX and Rx monitor will detect the IDLE sequence on their respective lanes and the information will be updated in a common transaction, the handle of which is being used by both the transmit and receive side monitors. Based on this, the monitor which detects that a change in the IDLE sequence is required on its side, will wait for a certain period and will perform the IDLE detection again to check if the change has happened or not. The IDLE sequence detection will happen by counting the number of K characters and number of M characters to reach a configured value and after that, if K count is higher than the M count, then IDLE1 sequence is detected, or else IDLE2 is detected. The configuration for K count and M count should be done based on the RTL implementation, so that monitors will also perform the IDLE detection at the same time. IDLE detection mechanism for BRC3 is not required as it will operate in IDLE3 only. Once the IDLE sequence is detected, the appropriate information is set in the particular mon-

itor instance's common transaction, so that the initialization state machine can access it to move the model's state.

The monitor instance of this block will continuously collect IDLE sequence from the line and will perform most of the required checks on the received sequence. The idle sequence collecting logic is same for all the modes, whereas different methods are executed for the checks, based on which IDLE sequence is being received.

3.6.5.5 PL State Machines

This block implements all the physical layer related state machines namely align, Initialization. It also implements other BRC3 specific state machines such as link retraining, Tx and Rx width state machines. The state machine variables are again updated in the common transaction and is used by all the blocks present inside the particular monitor component.

The monitor instance of this block will update appropriate status register fields based on the configured port number. It also performs asymmetric protocol checks on the transmitted and received width change command and its status.

3.6.5.6 Protocol Checker

The srio transaction which was created after destriping is used by the protocol checker to perform all the checks listed in the table ["Physical Layer Checks" on page 91]. The methods inside this class will update the common transaction, the handle of which, will be available in both Tx and Rx monitor. Thus, through this common transaction fields, link protocol information are exchanged between the Tx and Rx monitors and appropriate checks are performed. After the checks, the srio transaction is written into the monitor's analysis port, so that the subscribers which are connected to the monitor can take the srio transaction and perform its implemented functionality. The same will be passed on to the monitor present in the higher layer agents as well. This block will completely track the expected acknowledgement control symbols for the received packet, error recovery, retry recovery, crc checks, status control symbol frequency check, multiple vc check, and also link timeout checks. Though protocol only checks the link timeout for packet acknowledgement and link response control symbols, this block will perform the timeout check for all the expected control symbols that are scheduled by it.

Once a packet is received, this block will hold it in a queue until the acknowledgement for corresponding packet is received. It writes the packet to the analysis port for the upper layer monitors to process it only if packet-accepted control symbol is received for any packet. By this way the upper layer monitor will not process a packet more than once if it is either retried or not-accepted.

This block also updates its respective status register fields based on the configured port number, and also updates the error rate management related register fields when appropriate error is detected.

4 Sequences

4.1 Logical Layer Sequences

Table 12, “Logical Layer Sequences,” on page 63 shows the logical layer sequences.

Table 12: Logical Layer Sequences

S.No	Sequence	Description
1	srio_ll_nread_req_seq	Creates random nread request transactions
2	srio_ll_nwrite_req_seq	Creates random nwrite request transactions
3	srio_ll_nwrite_r_req_seq	Creates random nwrite_r request transactions
4	srio_ll_swrite_req_seq	Creates random swrite request transactions
5	srio_ll_atomic_inc_seq	Creates random Atomic Increment transactions
6	srio_ll_atomic_dec_seq	Creates random Atomic Decrement transactions
7	srio_ll_atomic_set_seq	Creates random Atomic Set transactions
8	srio_ll_atomic_clr_seq	Creates random Atomic Clear transactions
9	srio_ll_atomic_swap_seq	Creates random Atomic swap transactions
10	srio_ll_atomic_test_and_swap_seq	Creates random Atomic Test-and-Swap transactions
11	srio_ll_compare_and_swap_seq	Creates random Atomic Compare-and-Swap transactions
12	srio_ll_maintenance_rd_req_seq	Creates random Maintenance Read Request transactions
13	srio_ll_maintenance_wr_req_seq	Creates random Maintenance Write Request transactions
14	srio_ll_maintenance_port_wr_req_seq	Creates random Maintenance Port Write Request transactions
15	srio_ll_maintenance_rd_resp_seq	Creates random Maintenance Read Response transactions
16	srio_ll_io_rdsize_wdptr_err_seq	Creates SRIO IO Operations with rdsize_wdptr Error
17	srio_ll_io_wrsize_wdptr_err_seq	Creates SRIO IO Operations with wrsize_wdptr Error transaction
18	srio_ll_atomic_payload_err_seq	Creates Unsupported payload bytes such as 3,5,6,7 Bytes of transaction

S.No	Sequence	Description
19	srio_ll_swrite_payload_error_seq	Creates SWRITE - Payload less than Double Word
20	srio_ll_maintenance_wr_resp_seq	Creates random Maintenance Write Response transaction
21	srio_ll_resp_with_payload_seq	Creates random Logical Response with payload transaction and covers possible status values.
22	srio_ll_resp_without_payload_seq	Creates random Logical Response without payload transaction and covers possible status values.
23	srio_ll_message_req_seq	Creates random Data Message Request transaction (includes both single segment and multi segment)
24	srio_ll_msg_sseg_req_seq	Creates Data Message - Single Segment ie. Message size should be equal to segment size.
25	srio_ll_msg_mseg_req_seq	Creates Data Message - Multi Segment ie. Message size should be greater than segment size.
26	srio_ll_msg_interleaved_req_seq	Creates Data Message - Interleaved Data Message with different mailbox and letter
27	srio_ll_msg_inorder_resp_seq	Creates Data Message - Response with Done (In-order)
28	srio_ll_msg_outoforder_resp_seq	Data Message - Response with Done (Out-of-order)
29	srio_ll_msg_error_pkt_seq	Creates Data Message - Error Packet - Invalid Segment Size and Message length(ie.,greter than 4K)
30	srio_ll_db_error_pkt_seq	Creates Doorbell Message - with Payload
31	srio_ll_message_resp_seq	Creates random Data Message Response Sequence (Response generated based on input configuration)
32	srio_ll_doorbell_req_seq	Creates random Doorbell Request transaction
33	srio_ll_doorbell_resp_seq	Creates random Doorbell Response transaction
34	srio_ll_ds_pdu_seq	Creates random Data Streaming transaction (includes both single and multi segment stream transaction)

S.No	Sequence	Description
35	srio_ll_ds_sseg_req_seq	Creates Data Streaming -- Single Segment stream. ie., pdulength should be less than or equal to mtu size. Maximum payload is 256B
36	srio_ll_ds_mseg_req_seq	Creates Data Streaming -- Multi Segment stream. ie., pdulength should be greater than mtu size maximum payload is 64K
37	srio_ll_ds_error_req_seq	Creates Data Streaming -- Error Packet (includes sop = 0 for 1st segment eop = 0 for end segment odd= 1 for even number of half words pad =0,Needs pad byte to make it to half word)
38	srio_ll_ds_max_req_seq	Creates Data Streaming -- Maximum PDU and MTU Check -- Payload is 64K and MTU is 256 Bytes.
39	srio_ll_ds_err_seg_req_seq	Creates Data Streaming -- ERROR packet --start segment , continuous segment End segment is greater than MTU size
40	srio_ll_ds_interleaved_req_seq	Creates Data Streaming -- Interleaved Stream packet with different stream ID
41	srio_ll_ds_abort_req_seq	Creates Data Streaming -- Abort packet -- DS packet with no data payload
42	srio_ll_ds_pdumismatch_err_seq	Creates Data Streaming -- PDU Length error at End segment
43	srio_ll_ds_los_error_seq	Creates Data Streaming -- Loss of segment (continuous or end segment received) for closed context and (start segment received) for open context
44	srio_ll_ds_traffic_mgmt_seq	Creates random Data Streaming Traffic Management transaction
45	srio_ll_lfc_flow_arb_spdu_seq	Creates LFC - Flow arbitration with single PDU
46	srio_ll_lfc_flow_arb_mpdu_seq	Creates LFC - Flow arbitration with Multiple PDU
47	srio_ll_lfc_orphaned_xoff_seq	Creates LFC - without XON
48	srio_ll_lfc_multi_xoff_xon_same_flowid_seq	Creates LFC - Same FlowID with Multiple XOFF and XON Count
49	srio_ll_lfc_multi_xoff_xon_diff_flowid_seq	Creates LFC - Different FlowID with Multiple XOFF and XON Count

S.No	Sequence	Description
50	srio_ll_lfc_illegal_prio_req_seq	Creates LFC - Illegal priority (2'b11) request transaction.
51	srio_ll_lfc_illegal_resp_seq	Creates LFC - Illegal Priority(2'b00) response transaction
52	srio_ll_lfc_xoff_xon_seq	Creates LFC - Normal XOFF and XON Sequence
53	srio_ll_lfc_xon_without_xoff_seq	Creates LFC - Without XOFF Only XON
54	srio_ll_lfc_timeout_check_seq	Creates LFC - XOFF Count > XON, XON Count > XOFF
55	srio_ll_lfc_with_diff_id_seq	Creates LFC - XOFF with different SrcID and TargetID transaction
56	srio_ll_lfc_with_diff_flowid_seq	Creates LFC - XOFF with different flowID - Priority and CRF transaction
57	srio_ll_lfc_multi_orphaned_xoff_seq	Creates LFC - Multiple Orphaned XOFF transaction
58	srio_ll_gsm_rd_owner_seq	Creates random GSM Read Owner transaction
59	srio_ll_gsm_rd_to_own_owner_seq	Creates random GSM Read to Own Owner transaction
60	srio_ll_gsm_io_rd_owner_seq	Creates random GSM IO Read Owner transaction
61	srio_ll_gsm_rd_home_seq	Creates random GSM Read Home transaction
62	srio_ll_gsm_rd_to_own_home_seq	Creates random GSM Read to Own Home transaction
63	srio_ll_gsm_io_rd_home_seq	Creates random GSM IO Read Home transaction
64	srio_ll_gsm_dkill_home_seq	Creates random GSM DKill Home transaction
65	srio_ll_gsm_ikill_home_seq	Creates random GSM IKill Home transaction
66	srio_ll_gsm_tlbie_seq	Creates random GSM TLBIE transaction
67	srio_ll_gsm_tlbsync_seq	Creates random GSM TLBSYNC transaction
68	srio_ll_gsm_iread_home_seq	Creates random GSM IRead Home transaction
69	srio_ll_gsm_ikill_sharer_seq	Creates random GSM IKill Sharer transaction

S.No	Sequence	Description
70	srio_ll_gsm_dkill_sharer_seq	Creates random GSM DKill Sharer transaction
71	srio_ll_gsm_castout_seq	Creates random GSM CASTOUT transaction
72	srio_ll_gsm_flush_with_data_seq	Creates random GSM FLUSH with Data transaction
73	srio_ll_gsm_flush_without_data_seq	Creates random GSM FLUSH without Data transaction
74	srio_ll_invalid_ftype_seq	Creates Logical Layer packet with Invalid FTYPE
75	srio_ll_rand_all_packets_seq	Generates all types of packets.
76	srio_ll_rand_all_packets_err_seq	Generates all types of packets with errors injected.
77	srio_ll_pl_seq	Generates stimulus to both Physical and Logical Layers
78	srio_ll_ds_pdu_error_seq	Creates Data Streaming packets with invalid PDU
79	srio_ll_ds_mtu_error_seq	Creates Data Streaming packets with invalid MTU size
80	srio_ll_ds_sop_error_seq	Creates Data Streaming packets with invalid SOP size
81	srio_ll_ds_eop_error_seq	Creates Data Streaming packets with invalid EOP size
82	srio_ll_ds_odd_error_seq	Creates Data Streaming packets with invalid ODD size
83	srio_ll_ds_pad_error_seq	Creates Data Streaming packets with invalid PAD size
84	srio_ll_ttype_error_seq	Creates random TTYPE error transaction
85	srio_ll_resp_pri_error_seq	Creates random illegal response priority transaction
86	srio_ll_size_error_seq	Creates random packet with size exceed error transaction
87	srio_ll_payload_exist_error_seq	Creates random packet with payload exist error transaction
88	srio_ll_doubleword_align_error_seq	Creates random packet with double word alignment error transaction
89	srio_ll_lfc_pri_error_seq	Creates random LFC – illegal priority sequence

S.No	Sequence	Description
90	srio_ll_msg_size_error_seq	Creates random Message packet with illegal size error sequence
91	srio_ll_msgseg_error_seq	Creates random message packet with invalid segment size error sequence.
92	srio_ll_lfc_user_gen_xoff_seq	Creates LFC user generated xoff sequence
93	srio_ll_lfc_user_gen_xon_seq	Creates LFC user generated xon sequence
94	srio_ll_io_random_seq	Creates IO random sequence
95	srio_ll_ds_mseg_single_mtu_seq	Creates multi segment DS packet with single MTU value sequence
96	srio_ll_port_resp_timeout_reg_seq	Creates transaction to configure Port Response Timeout Register
97	srio_ll_all_atomic_req_seq	Creates Maintenance read-write packet to configure Port Response Timeout Register
98	srio_ll_ds_concurrent_seq	Creates DS packet with valid MTU and PDU length value
99	srio_ll_ds_max_seg_support_seq	Creates DS packets to support maximum segment sequence
100	srio_ll_ds_mtu_reserved_seq	Creates DS packet with invalid MTU value sequence
101	srio_ll_ds_s_e_err_seq	Creates DS packet with invalid Start and End bit sequence
102	srio_ll_ds_traffic_seq	Creates DS packet with user defined COS and StreamID value sequence
103	srio_ll_ds_traffic_mgmt_basic_stream_xoff_seq	Creates Traffic Management basic stream xoff packet sequence
104	srio_ll_ds_traffic_mgmt_basic_stream_xon_seq	Creates Traffic Management basic stream xon packet sequence
105	srio_ll_normal_ds_payload_size_err_seq	Creates DS payload size Error packet sequence
106	srio_ll_ds_traffic_mgmt_xtype_err_seq	Creates Traffic Management packet with invalid xtype sequence
107	srio_ll_ds_traffic_mgmt_user_rate_xoff_seq	Creates user generated Traffic Management xoff packet for rate mode sequence
108	srio_ll_ds_traffic_mgmt_user_rate_xon_seq	Creates user generated Traffic Management xon packet for rate mode sequence
109	srio_ll_ds_traffic_mgmt_user_credit_xon_seq	Creates user generated Traffic Management xon packet for credit mode sequence
110	srio_ll_ds_traffic_mgmt_user_credit_xoff_seq	Creates user generated Traffic Management xoff packet for credit mode sequence

S.No	Sequence	Description
111	srio_ll_ds_traffic_mgmt_tmop_err_seq	Creates Traffic Management packet with invalid TMOP value sequence
112	srio_ll_ds_traffic_mgmt_parameter1_err_seq	Creates Traffic Management packet with invalid parameter1 value sequence
113	srio_ll_ftype_default_seq	Creates packets with random ftype values sequence
114	srio_ll_lfc_unsupport_flowid_seq	Creates LFC packet with unsupported-flowID value sequence
115	srio_ll_maintenance_wr_rd_seq	Create maintenance read-write packets to configure all registers sequence
116	srio_ll_lfc_xon_arb_0_seq	Creates LFC flow arbitration XON sequence number 0
117	srio_ll_lfc_xon_arb_1_seq	Create LFC flow arbitration XON sequence number 1
118	srio_ll_lfc_ds_single_pdu_arb_seq	Creates FAM DS single PDU sequence
119	srio_ll_lfc_ds_multi_pdu_arb_seq	Creates FAM DS multi PDU sequence
120	srio_ll_lfc_request_flow_spdu_1_seq	Creates FAM request for single PDU sequence number 0
121	srio_ll_lfc_request_flow_spdu_0_seq	Creates FAM request for single PDU sequence number 0
122	srio_ll_lfc_release_0_seq	Creates FAM release for sequence number 0
123	srio_ll_lfc_release_1_seq	Creates FAM release for sequence for 1
124	srio_ll_flow_arb_support_reg_seq	Creates FAM support enable sequence
125	srio_ll_lfc_request_flow_mpdu_1_seq	Creates FAM request for multi PDU sequence number 1
126	srio_ll_lfc_request_flow_mpdu_0_seq	Creates FAM request for multi PDU sequence number 0
127	srio_ll_traffic_mgmt_tm_type_mode_err_seq	Creates Traffic Management packet with invalid TM_Mode value sequence
128	srio_ll_ds_traffic_mgmt_mask_err_seq	Creates Traffic Management packet with invalid mask value sequence
129	srio_ll_ds_traffic_mgmt_diff_operation_seq	Creates Traffic Management packet for different operation sequence
130	srio_ll_ds_traffic_mgmt_user_credit_err_seq	Creates user generated Traffic Management packet with credit support and invalid parameter values sequence
131	srio_ll_ds_traffic_mgmt_specific_stream_xoff_seq	Creates Traffic Management packet for specific Stream xoff sequence

S.No	Sequence	Description
132	srio_ll_ds_traffic_mgmt_specific_stream_xon_seq	Creates Traffic Management packet for specific Stream xon sequence
133	srio_ll_ds_traffic_mgmt_specific_cos_xoff_seq	Creates Traffic Management packet for specific COS xoff sequence
134	srio_ll_ds_traffic_mgmt_specific_cos_xon_seq	Creates Traffic Management packet for specific COS xon sequence
135	srio_ll_ds_traffic_mgmt_group_of_cos_xoff_seq	Creates Traffic Management packet for Group of COS xoff sequence
136	srio_ll_ds_traffic_mgmt_group_of_cos_xon_seq	Creates Traffic Management packet for Group of COS xon sequence
137	srio_ll_ds_traffic_mgmt_random_cos_xoff_seq	Creates Traffic Management packet for random of COS xoff sequence
138	srio_ll_ds_traffic_mgmt_random_cos_xon_seq	Creates Traffic Management packet for random of COS xon sequence
139	srio_ll_lfc_ds_seq	Creates DS packet with valid Priority and Crf value sequence
140	srio_ll_ds_all_traffic_xoff_seq	Creates Traffic Management packet for all traffic xoff sequence
141	srio_ll_ds_all_traffic_xon_seq	Creates Traffic Management packet for all traffic xon sequence
142	srio_ll_nwrite_nread_34_addr_seq	Creates NWRITE and NREAD with 34 bits addressing bit packets sequence
143	srio_ll_nwrite_nread_50_addr_seq	Creates NWRITE and NREAD with 50 bits addressing bit packets sequence
144	srio_ll_nwrite_nread_66_addr_seq	Creates NWRITE and NREAD with 66 bits addressing bit packets sequence
145	srio_ll_lfc_ds_random_prio_seq	Creates DS packet with random priority sequence
146	srio_ll_nwrite_nread_mem_access_seq	Creates nwrite and nread packets to access memory block sequence
147	srio_ll_user_gen_random_prio_seq	Creates user generated random priority sequence
148	srio_ll_maintenance_ds_support_reg_seq	Creates Maintenance read-write packet to configure for DS packet sequence
149	srio_ll_ds_traffic_random_streamid_seq	Creates DS packet with random StreamID sequence
150	srio_ll_ds_traffic_random_streamid_cos_seq	Creates DS packet with random StreamID and COS sequence

S.No	Sequence	Description
151	srio_ll_ds_all_traffic_seq	Creates Traffic Management packed with wild card 3'b111 sequence
152	srio_ll_ds_all_traffic_mgmt_credit_control_seq	Creates Traffic Management packet with credit control sequence
153	srio_ll_illegal_io_seq	Creates IO packets with illegal ttype,rd_size and wr_size sequence
154	srio_ll_illegal_gsm_seq	Creates GSM packets with illegal ttype,rd_size and wr_size sequence
155	srio_ll_illegal_msg_seq	Creates MSG packets with illegal ttype and segments sequence
156	srio_ll_vc_lfc_xon_seq	Creates LFC xon packet with multi VC support sequence
157	srio_ll_vc_lfc_xoff_seq	Creates LFC xon packet with multi VC support sequence
158	srio_ll_vc_ds_mseg_req_seq	Creates DS multi segments packet with multi VC support sequence
159	srio_ll_multi_vc_nwrite_swrite_seq	Creates nwrite and swrite packet with multi VC support sequence
160	srio_ll_msg_mseg_req_with_err_seq	Creates message packets with invalid length sequence
161	srio_ll_msg_mseg_max_req_seq	Creates message packets with maximum length sequence
162	srio_ll_ds_traffic_mgmt_random_basic_stream_seq	Creates DS Traffic Management Basic packet with random parameter and specific stream values sequence
163	srio_ll_ds_traffic_mgmt_random_rate_stream_seq	Creates DS Traffic Management Rate packets with random parameter and specific stream values sequence
164	srio_ll_ds_traffic_mgmt_random_credit_stream_seq	Creates DS Traffic Management Credit packets with random parameter and specific stream values sequence
165	srio_ll_atomic_invalid_size_seq	Creates Atomic packets with illegal size sequence
166	srio_ll_ds_mseg_req_err_seq	Creates DS multi segment packets with invalid size sequence
167	srio_ll_ds_max_streamid_seq	Creates DS packets with maximum Stream_ID sequence

S.No	Sequence	Description
168	srio_ll_msg_unsupported_seq	Creates message packets for unsupported message transition sequence
169	srio_ll_db_unsupported_seq	Creates message packets for unsupported doorbell transition sequence
170	srio_ll_outstanding_unack_req_seq	Creates random packets request with maximum unacknowledged request sequence
171	srio_ll_unexp_msg_resp_req_seq	Creates unexpected message response packets without message request packet sequence
172	srio_ll_msg_mseg_max_mbox_letter_req_seq	Creates message packets with maximum letter and fixed mbox values sequence
173	srio_ll_req_seq	Creates nread request packets for illegal ftype corruption in callback sequence
174	srio_ll_gsm_pkt_for_illegal_resp_req_seq	Creates GSM packets for illegal response sequence
175	srio_ll_msg_mseg_req_cov_seq	Creates multi segment message with all possible ssize, letter, mbox, message length sequence
176	srio_ll_pkt_invalid_tt_seq	Creates packets with invalid TT values sequence
177	srio_ll_vc_user_gen_seq	Creates user generated packets with multi VC support sequence
178	srio_ll_vc_lfc_user_gen_xoff_seq	Creates LFC XOFF packets with VC support sequence
179	srio_ll_vc_lfc_user_gen_xon_seq	Creates LFC XON packets with VC support sequence

4.2 Transport Layer Sequences

Table 13, “Transport Layer Sequences,” on page 72 shows the transport layer sequences.

Table 13: Transport Layer Sequences

S.No	Sequence	Description
1	srio_tl_pkt_seq	Transport Layer Packet Sequence
2	srio_tl_pkt_err_seq	Transport Layer Packet Sequence with error (eg., corrupt tt field)
3	srio_tl_rand_all_packets_seq	Generates all types of packets.
4	srio_tl_rand_all_packets_err_seq	Generates all types of packets with errors injected.

4.3 Physical Layer Sequences

Table 14, “Physical Layer Sequences,” on page 73 shows the physical layer sequences.

Table 14: Physical Layer Sequences

S.No	Sequence	Description
1	srio_pl_pkt_acc_cs_seq	Packet Accepted CS Sequence
2	srio_pl_pkt_rty_cs_seq	Packet Retry CS Sequence
3	srio_pl_pkt_na_cs_seq	Packet Not Accepted CS Sequence
4	srio_pl_status_cs_seq	Status CS Sequence
5	srio_pl_vc_st_cs_seq	VC Status CS Sequence
6	srio_pl_link_res_cs_seq	Link Request CS Sequence
7	srio_pl_sop_cs_seq	Start of Packet CS Sequence
8	srio_pl_sop_padded_cs_seq	Start of Packet padded CS sequence
9	srio_pl_sop_unpadded_cs_seq	Start of Packet unpadded CS sequence
10	srio_pl_stomp_cs_seq	Stomp CS Sequence
11	srio_pl_eop_cs_seq	End of Packet CS Sequence
12	srio_pl_eop_padded_cs_seq	End of Packet padded CS Sequence
13	srio_pl_eop_unpadded_cs_seq	End of Packet Unpadded CS Sequence
14	srio_pl_restart_rty_cs_seq	Restart from Retry CS Sequence
15	srio_pl_link_req_cs_seq	Link Request CS Sequence
16	srio_pl_multicast_cs_seq	Multicast CS Sequence
17	srio_pl_nop_cs_seq	NOP CS Sequence
18	srio_pl_link_req_rst_dev_cs_seq	Link request Reset Device CS Sequence
19	srio_pl_link_req_rst_port_cs_seq	Link Request Reset Port CS Sequence
20	srio_pl_link_req_port_status_cs_seq	Link Request Port Status CS Sequence
21	srio_pl_ip_st_cs_seq	Input Status CS Sequence
22	srio_pl_timestamp_cs_seq	Timestamp CS Sequence
23	srio_pl_loop_resp_cs_seq	Loop Response CS Sequence
24	srio_pl_voq_bp_cs_seq	VoQ Back pressure CS Sequence
25	srio_timing_cs_seq	Timing CS Sequence
26	srio_descr_sync_break_cs_seq	Descrambler sync break CS sequence
27	srio_pl_ackid_err_cs_seq	Ackid error CS sequence
28	srio_pl_crc_err_cs_seq	Crc error CS sequence
29	srio_pl_loss_descr_sync_cs_seq	Loss of descrambler sync CS sequence
30	srio_pl_invalid_char_cs_seq	Invalid/Illegal character CS sequence

S.No	Sequence	Description
31	srio_pl_pkt_na_lack_buf_res_cs_seq	Packet Not Accepted due to lack of buffer resources CS sequence
32	srio_pl_pkt_na_ackid_status_cs_seq	Packet Not Accepted ackid status CS sequence
33	srio_pl_pkt_ackid_error_seq	Packet Ackid Error Sequence
34	srio_pl_pkt_early_crc_err_seq	Packet early crc error sequence
35	srio_pl_pkt_final_crc_err_seq	Packet final crc error sequence
36	srio_pl_pkt_illegal_prio_err_seq	Packet illegal priority error sequence
37	srio_pl_pkt_illegal_crf_err_seq	Packet illegal crf error sequence
38	srio_pl_pkt_size_err_seq	Packet size greater than the allowed maximum packet size error sequence
39	srio_pl_stomp_pkt_cancel_seq	Cancelling a packet by a Stomp sequence
40	srio_pl_restart_retry_pkt_cancel_seq	Cancelling a packet by restart from retry sequence
41	srio_pl_link_request_pkt_cancel_seq	Cancelling a packet by a link request sequence
42	srio_pl_sync_break_seq	Synchronization break sequence
43	srio_pl_align_break_seq	Alignment break sequence
44	srio_pl_idle2_csfield_corruption_seq	Idle2 csfield corruption sequence
45	srio_pl_idle2_psr_corruption_seq	Idle2 pseudo random field corruption sequence
46	srio_pl_idle2_csmarker_corruption_seq	Idle2 cs field marker corruption sequence
47	srio_pl_idle2_descr_sync_break_seq	Idle2 descrambler sync break sequence
48	srio_pl_idle2_csfield_truncation_seq	Idle2 cs field truncation sequence
49	srio_pl_idle2_psr_truncation_seq	Idle2 pseudo random truncation sequence
50	srio_pl_idle2_csmarker_truncation_seq	Idle2 cs marker truncation sequence
51	srio_pl_idle2_csfield_update_seq	Idle2 cs field update sequence
52	srio_pl_force1x_mode_portwidth_override_seq	Force 1x mode port width override sequence
53	srio_pl_force1x_mode_laner_portwidth_override_seq	Force 1x mode lane R port width override sequence
54	srio_pl_nxmode_enabled_2x_disabled_portwidth_override_seq	Force Nx mode enabled 2x mode disabled port width override sequence
55	srio_pl_2xmode_enabled_nx_disabled_portwidth_override_seq	Force 2x mode enabled Nx disabled port width override sequence

S.No	Sequence	Description
56	srio_pl_force_reinit_seq	Force re-initialization sequence
57	srio_pl_dis_nxmode_sm_seq	Discovery to Nxmode state transition sequence
58	srio_pl_dis_2xmode_sm_seq	Discovery to 2xmode state transition sequence
59	srio_pl_dis_1xmode_ln0_sm_seq	Discovery to 1xmode_lane0 state transition sequence
60	srio_pl_dis_1xmode_ln1_sm_seq	Discovery to 1xmode_lane1 state transition sequence
61	srio_pl_dis_1xmode_ln2_sm_seq	Discovery to 1xmode_lane2 state transition sequence
62	srio_pl_dis_sl_sm_seq	Discovery to Silent state transition sequence
63	srio_pl_nxmode_sl_sm_seq	Nxmode to Silent state transition sequence
64	srio_pl_nxmode_dis_sm_seq	Nxmode to Discovery state transition sequence
65	srio_pl_2xmode_sl_sm_seq	2xmode to Silent state transition sequence
66	srio_pl_2xmode_2x_rec_sm_seq	2xmode to 2x recovery state transition sequence
67	srio_pl_1xmode_ln0_sl_sm_seq	1xmode lane0 to Silent state transition sequence
68	srio_pl_1xmode_ln0_1x_rec_sm_seq	1xmode lane0 to 1x recovery state transition sequence
69	srio_pl_1xmode_ln1_sl_sm_seq	1xmode lane1 to Silent state transition sequence
70	srio_pl_1xmode_ln1_1x_rec_sm_seq	1xmode lane1 to 1x recovery state transition sequence
71	srio_pl_1xmode_ln2_sl_sm_seq	1xmode lane2 to Silent state transition sequence
72	srio_pl_1xmode_ln2_1x_rec_sm_seq	1xmode lane2 to 1x recovery state transition sequence
73	srio_pl_2x_rec_2xmode_sm_seq	2x Recovery to 2xmode state transition sequence

S.No	Sequence	Description
74	srio_pl_x_rec_1xmode_ln0_sm_seq	2x Recovery to 1xmode lane0 state transition sequence
75	srio_pl_2x_rec_1xmode_ln1_sm_seq	2x Recovery to 1xmode lane1 state transition sequence
76	srio_pl_2x_rec_sl_sm_seq	2x Recovery to silent state transition sequence
77	srio_pl_1x_rec_sl_sm_seq	1x Recovery to Silent state transition sequence
78	srio_pl_1x_rec_1xmode_ln0_sm_seq	1x Recovery to 1xmode lane0 state transition sequence
79	srio_pl_1x_rec_1xmode_ln1_sm_seq	1x Recovery to 1xmode lane1 state transition sequence
80	srio_pl_1x_rec_1xmode_ln2_sm_seq	1x Recovery to 1xmode lane2 state transition sequence
81	srio_pl_aymmetry_silent_sm_seq	Asymmetry to Silent state transistion sequence
82	srio_pl_1x_rec_1x_retrain_sm_seq	1x Recovery to 1x Retrain state transition sequence
83	srio_pl_1x_retrain_1x_rec_sm_seq	1x Retrain to 1x Recovery state transition sequence
84	srio_pl_nx_rec_nx_retrain_sm_seq	Nx Recovery to Nx Retrain state transition sequence
85	srio_pl_nx_retrain_nx_rec_sm_seq	Nx Retrain to Nx Recovery state transition sequence
86	srio_pl_2x_rec_2x_retrain_sm_seq	2x Recovery to 2x Retrain state transition sequence
87	srio_pl_2x_retrain_2x_rec_sm_seq	2x Retrain to 2x Recovery state transition sequence
88	srio_pl_nxm_nxr_sm_seq	Nx Mode to Nx Recovery state transition sequence
89	srio_pl_nxr_nxm_sm_seq	Nx Recovery to Nx Mode state transition sequence
90	srio_pl_nxr_1xm0_sm_seq	Nx Recovery to 1x Mode0 state transition sequence
91	srio_pl_nxr_1xm1_sm_seq	Nx Recovery to 1x Mode1 state transition sequence
92	srio_pl_nxr_1xm2_sm_seq	Nx Recovery to 1x Mode2 state transition sequence

S.No	Sequence	Description
93	srio_pl_nxr_sil_sm_seq	Nx Recovery to Silent Mode state transition sequence
94	srio_pl_nxm_asymetry_sm_seq	Nx Mode to Asymmetry Mode state transition sequence
95	srio_pl_2xm_asymetry_sm_seq	2x Mode to Asymmetry Mode state transition sequence
96	srio_pl_nxr_2xm_sm_seq	Nx Recovery to 2x Mode state transition sequence
97	srio_pl_rand_state_trans_seq	Random state transition sequence
98	srio_pl_rand_all_packets_seq	Generates all types of packets.
99	srio_pl_rand_all_packets_err_seq	Generates all types of packets with errors injected.
100	srio_pl_nx_mode_support_disable_seq	Creates NX mode support disable sequence
101	srio_pl_x2_mode_support_disable_seq	Creates 2X mode support disable sequence
102	srio_pl_pkt_na_ackid_err_cs_seq	Creates packet not accepted ackID error sequence
103	srio_pl_pkt_na_crc_err_cs_seq	Creates packet not accepted CRC error sequence
104	srio_pl_pkt_na_non_maintenace_rep_stop_cs_seq	Creates packet not accepted non maintenance error sequence
105	srio_pl_pkt_na_invalid_char_cs_seq	Creates packet not accepted invalid character error sequence
106	srio_pl_pkt_na_lack_buf_res_cs_seq	Creates packet not accepted lack of buffer error sequence
107	srio_pl_pkt_na_loss_descr_sync_cs_seq	Creates packet not accepted loss of desr error sequence
108	srio_pl_nwrite_swrite_req_seq	Creates nwrite and swrite sequence and force it directly to pl sequencer
109	srio_pl_ll_io_random_seq	Creates IO packets with randomized pl variables sequence
110	srio_txrx_seq	Creates packets with TxRx status control symbols sequence
111	srio_pl_sop_nwrite_eop_seq	Creates packet with sop ,nwrite packets and eop control symbol sequence
112	srio_pl_gen3_sop_padded_seq	Creates GEN3 packets with padded start of packet (sop) sequence
113	srio_pl_gen3_eop_padded_seq	Creates GEN3 packets with padded end of packets (eop) sequence

5 Sequence Item

SRIO VIP's sequence item is named as srio_trans and it contains the fields of packets/control symbols defined in the serial RapidIO specification. It also includes miscellaneous fields and random constraints. pTable 15, "SRIO VIP's Sequence Item," on page 78 provides the description of various fields defined in this object.

Table 15: SRIO VIP's Sequence Item

S.No	Field Name	Size	Description
Common Fields			
1	transaction_kind	srio_trans_kind	Transaction Kind {SRIO_PACKET,SRIO_CS,SRIO_STATE_MC}
Logical Layer Fields			
2	ftype	4-bits	Format type
3	wdptr	1-bit	Word pointer, used in conjunction with the data size (rdsize and wrsize) fields
4	rdsize	4-bits	Data size for read transactions, used in conjunction with the word pointer (wdptr) bit
5	wrsize	4-bits	Write data size for sub-double-word transactions, used in conjunction with the word pointer (wdptr) bit
6	SrcTID	8-bits	The packet's transaction ID
7	ttype	4-bits	Transaction Type
8	address	29-bits	Physical address
9	ext_address	32-bits	Specifies the most significant 16 bits of a 50-bit physical address or 32 bits of a 66-bit physical address.
10	flowID	3-bits	Includes Priority and CRF
11	xamsbs	2-bits	Extended address most significant bits. Further extends the address specified by the address and extended address fields by 2 bits.
12	config_offset	21-bits	Double-word offset into the CAR/CSR register block for reads and writes
13	info_lsb	8-bits	Software-defined information field LSB
14	info_msb	8-bits	Software-defined information field MSB
15	msg_len	4-bits	Total number of packets comprising the message operation.
16	ssize	4-bits	Standard message packet data size.
17	letter	2-bits	Identifies a slot within a mailbox.
18	mbox	2-bits	Specifies the recipient mailbox in the target element

S.No	Field Name	Size	Description
19	msgseg_xmbox	4-bits	msg_seg -> For multiple packet data message operations, specifies the part of the message supplied by the packet. xmbox -> For single packet data message operations, specifies the upper 4 bits of the mailbox targeted by the packet.
20	cos	8-bits	class of service
21	S	1-bit	Start - If set, this packet is the first segment of a new PDU
22	E	1-bit	End - If set, this packet is the last segment of a PDU
23	xh	1-bit	Extended header - is used for traffic management.
24	odd	1-bit	Odd - If set, the data payload has an odd number of half-words
25	pad	1-bit	Pad - If set, a pad byte was used to pad to a half-word boundary
26	StreamID	16-bits	Traffic stream identifier
27	pdulength	16-bits	PDU length
28	Xtype	3-bits	Traffic Management Packet
29	TMOP	4-bits	Indicates which type of Stream Management Message
30	wildcard	3-bits	Indicates VSID dest/class/stream wildcard
31	mask	8-bits	Class Mask: Used to mask portions of the class of service (COS) field to allow groups of classes to be included in the message
32	parameter1	8-bits	Parameter1: Argument specific to the TM message operation
33	parameter2	8-bits	Parameter2: Argument specific to the TM message operation
34	SecTID	8-bits	Original requestor's, or secondary, transaction ID for intervention
35	SecID	4-bits	Original requestor's, or secondary, ID for intervention
36	SecDomain	4-bits	Original requestor's, or secondary, do-main for intervention
37	targetID_Info	8-bits	Target Transaction ID / Target Information of Msg/ Doorbell Response
38	trans_status	4-bits	Type of RESPONSE status
39	payload[$]	8-bits	Payload Queue

S.No	Field Name	Size	Description
Transport Layer Fields			
40	SourceID	32-bits	Source Device ID
41	DestinationID	32-bits	Destination Device ID
42	tt	2-bits	Transport Type
43	hop_count	8-bits	Hop Count
Physical Layer Fields			
44	ackid	12-bits	Acknowledge ID.
45	gen3_ackid_msb	6-bits	MSB of ackid decoded from sop-padded or sop-unpadded control symbol.
46	sop	1-bit	Start of Packet.
47	eop	1-bit	End of Packet
48	prio	2-bits	Priority
49	crf	1-bit	Critical Request Flow
50	vc	1-bit	Virtual Channel
51	vcid	4-bits	Virtual Channel ID
52	early_crc	16-bits	Early crc
53	final_crc	16-bits	Final crc
54	crc_32	32-bits	Crc 32.Specific to Gen3.0
55	final_crc_err	1-bit	Final Crc Error
56	early_crc_err	1-bit	Early Crc Error
57	crc32_err	1-bit	Crc32 Error
58	cs_type	srio_pl_cs_type	CS Type {CS24,CS48,CS64}
59	cs_kind	srio_pl_cs_kind	CS Kind {SRIO_DELIM_SC,SRIO_DELIM_PD}
60	stype0	4 bits	Stype0
61	param0	12-bits	Parameter0
62	param1	12-bits	Parameter1
63	brc3_stype1_msb	2 bits	Used only for GEN3.0. This field concatenated with stype1 and cmd field gives the stype1 value for CS64.
64	stype1	3 bits	Stype1
65	cmd	3-bits	Command
66	cs_crc5	5-bits	Crc5.Crc5 for short CS24
67	cs_crc13	13-bits	Crc13. Crc13 for long CS48
68	cs_crc_24	24-bits	Crc24.Crc24 for gen3.0 CS64

S.No	Field Name	Size	Description
69	state	srio_pl_state_kind	State. State variable for holding the current state of initialization state machine {SILENT,SEEK,DISCOVERY,NX_MODE,2X_MODE,2X_RECOVERY,1X_MODE_LANE0,1X_MODE_LANE1,1X_MODE_LANE2,1X_RECOVERY}
70	next_state	srio_pl_state_kind	Next State.Next state variable for moving to the specific state of initialization state machine {SILENT,SEEK,DISCOVERY,NX_MODE,2X_MODE,2X_RECOVERY,1X_MODE_LANE0,1X_MODE_LANE1,1X_MODE_LANE2,1X_RECOVERY}
Miscellaneous Fields			
71	ll_err_kind	ll_err_kind	LL Error Kind NONE,MAX_SIZE_ERR,FTYPE_ERR,TTYPE_ERR,PAYLOAD_ERR,RESP_RSVD_STS_ERR,RESP_PRI_ERR,RESP_PAYLOAD_ERR,SIZE_ERR,NO_PAYLOAD_ERR,PAYLOAD_EXIST_ERR,ATAS_PAYLOAD_ERR,AS_PAYLOAD_ERR,ACAS_PAYLOAD_ERR,DW_ALIGN_ERR,LFC_PRI_ERR,DS_MTU_ERR,DS_PDU_ERR,DS_SOP_ERR,DS_EOP_ERR,DS_ODD_ERR,DS_PAD_ERR,MSG_SSIZE_ERR,MSGSEG_ERR,SRC_OP_UNSUPPORTED_ERR,DEST_OP_UNSUPPORTED_ERR,OUTSTANDING_REQ_ERR,OUTSTANDING_SEQNO_ERR,UNEXP_RESP_ERR,UNEXP_RESP_STS_ERR,HOP_COUNT_ERR,TM_BLOCKED_DS_ERR,LFC_BLOCKED_PKT_ERR,MISSING_DS_CONTEXT_ERR,DSSEG_ERR,REQ_TIMEOUT_ERR,RESP_TIMEOUT_ERR,VC_ERR,INVALID_PRI_ERR,INVALID_FLOWID_ERR,REQ_PIPELINING_ERR,IMPROPER_RELEASE_ERR,INVALID_FLOWARB_CMD_ERR,NONZERO_RESERVED_FLD_ERR,REG_CONFIG_ERR,RESERVED_MASK_ERR,RESERVED_PARAMETER_ERR,RESERVED_TMOP_ERR
72	tl_err_kind	tl_err_type	TL Error Type TL_NULL_ERR,DEST_ID_MISMATCH_ERR,UNSUPPORTED_TT_ERR,RESERVED_TT_ERR

S.No	Field Name	Size	Description
73	pl_err_kind	pl_err_kind	PL Error Kind NO_ERR,EARLY_CRC_ERR,FINAL_CRC_ERR ,ACKID_ERR,CS_FIELD_COR,PSR_COR,CSM ARKER_COR,DESC_SYNC_BREAK,CS_FIELD _TRU,PSR_TRU,CSMARKER_TRU,CSFIELD_ UPDATE.
74	payload_err	bool	Payload size greater than allowed
75	wdptr_rdsize_err	bool	Invalid wdptr and rdsize
76	wdptr_wrsize_err	bool	Invalid wdptr and wrsize
77	crc_err	bool	If TRUE,CRC error is created
78	stomp_err	bool	Indicate STOMP End during packet transmission
79	ackid_err	bool	If TRUE, wrong AckID is transmitted
80	cs_err	srio_pl_cs_er r_kind	CS Error {BAD_CHAR,CRC_ERR,DELIM_ERR,ALIGN_ ERR}
81	usr_gen_pkt	bit	If 1, indicates that the packet is generated by user. User has taken care of constructing the packet with the required values.
82	usr_directed_ll_re sponse_en	bool	If TRUE, user directed response type is sent. If FALSE, response type is selected from LL configu- ration parameters
83	usr_directed_ll_re ponse_type	srio_ll_resp_ kind	LL_NO_RESP,LL_DONE,LL_ERROR,LL_RETR Y
84	usr_directed_ll_re sponse_delay	integer	Decides the latency of a LL response packet trans- mitted.
85	usr_directed_pl_r esponse_en	bool	If TRUE, user configured delay and port status are inserted while PL responses are sent.
86	usr_directed_port _status	integer	User directed port status value for PL response packets.
87	usr_directed_pl_r esponse_delay	integer	Decides the latency of a PL response packet trans- mitted.
88	usr_directed_pl_a ck_en	bool	If TRUE, user directed response type is sent. If FALSE, response type is selected from PL configu- ration parameters.
89	usr_directed_pl_a ck_type	srio_pl_ack_ kind	PL_ACCEPT,PL_NOT_ACCEPT,PL_RETRY.
90	usr_directed_pl_n ac_cause	srio_pl_nac_ cause	User directed cause field if packet not accepted response is sent.

S.No	Field Name	Size	Description
91	usr_directed_pl_ack_delay	integer	Decides the latency of an acknowledge packet transmitted.
92	packet_gap	int	Delay between packets for transmission
93	msg_type	bit	Used by LL monitor to save the type of msg as single segment/multi segment
94	ll_err_detected	ll_err_kind	Type(ll_err_kind) of LL error detected by the LL monitor
95	tl_err_detected	tl_err_type	Type(tl_err_type) of TL error detected by the TL monitor
96	ll_err_encountered	bit	Set by LL monitor when any LL error is encountered
97	tl_err_encountered	bit	Set by TL monitor when any TL error is encountered
98	pl_err_encountered	bit	Set the protocol checker instance, so that the upper layer monitors need not process the error transaction. It will be used by the functional coverage instance.
99	cs_crc24_err	bit	Indicates crc24 error is detected in the received control symbol transaction.
100	do_pack	function	The function do_pack method acts as the user-definable hook called by the <pack> method and is used to override the actual uvm do_pack method to include its fields in a pack. This function packs the packet fields into bit stream. Input to function is srio_trans object and output is bit array.
101	do_unpack	function	The function do_unpack method acts as the user-definable hook called by the <unpack> method and is used to override the actual uvm do_unpack method to include its fields in an unpack operation. This function unpacks the bit stream into srio_trans object.
102	do_copy	function	The function do_copy method acts as the user-definable hook called by the <copy> method and is used to override the actual uvm do_copy method to include its fields in a copy operation. This function copies the fields from the current srio_trans object to new srio_trans object.

S.No	Field Name	Size	Description
103	do_compare	function	The function do_compare method acts as the user-definable hook called by the <compare> method and is used to override the actual uvm do_compare method to include its fields in a compare operation. Compares its fields with the received srio_trans object's fields.
104	do_print	function	The function do_print method acts as the user-definable hook called by the <print> and <sprint> method and is used to override the actual uvm do_print method to ensure a consistent output format. Prints the srio_trans object's fields.

6 List of Checks

6.1 Logical Layer Checks

The logical layer agent monitor checks are listed in Table 16, “Logical Layer Checks,” on page 85.

Table 16: Logical Layer Checks

S.No	Check Description	Specification Reference
Common LL Checks		
1	Received a request packet with an outstanding source TID	Part 1, Section 3.1.
2	Received write requests with data payload greater than max size (256 bytes)	Part 1, Section 4.1.2 Part 1, Section 4.1.8
3	Response received with reserved status field	Part 2, Section 4.3.1
4	Received a packet with invalid ftype	Part 1, Section 4.1
5	Received a packet with valid ftype and invalid ttype	Part 3, Section 2.5.3
6	Received a packet where the device does not have the support to process it	Part 1, Section 5.4.8 Part 2, Section 5.4.2 Part 5, Section 5.4.2 Part 10, Section 5.5.2
7	Transmitted a packet where the processing element does not have the support to issue it	Part 1, Section 5.4.7 Part 2, Section 5.4.1 Part 5, Section 5.4.1 Part 10, Section 5.5.1
8	Requests received with priority = 3	Part 6, Section 5.6.3

S.No	Check Description	Specification Reference
9	Response received with priority = 0	Part 6, Section 5.6.3
10	Response received with a priority not greater than request's priority	Part 6, Section 5.6.3
11	Response received for a non-outstanding request i.e. unsolicited response	Part 8, Section 2.5.3
TYPE 1		
12	Packets received with data payload	Part 5, Section 4.2.5
13	Received a reserved combination of rdsize and wdptr	Part 5, Section 4.2.5
TYPE 2 - Generic Checks		
14	Received packet with data payload	Part 1, Section 4.1.5
TYPE 2 - NREAD		
15	Received response to NREAD where the requested length and received data payload mismatches	Part 1, Section 3.3.1
TYPE 2 - ATOMIC		
16	Atomic transactions of sizes other than 1,2 and 4 bytes	Part 1, Section 3.3.4
17	Received packet with data payload greater than the expected number of bytes corresponding to{wdptr,wrsize}	Part 1, Section 4.1.7
TYPE 5 - Generic Checks		
18	Received a reserved combination of wrsize and wdptr	Part 1, Section 4.1.2
19	Received packet with data payload greater than the expected number of bytes corresponding to{wdptr,wrsize}	Part 1, Section 4.1.7
20	Received packets without data payload	Part 1, Section 4.1.7.
21	Received packet with data payload not matching the expected number of bytes, where the wrsize is lesser than or equal to single double word	Part 5, Section 4.2.8.
22	Data payload greater than the wrsize in multiple double-word transactions	Part 1, Section 4.1.2
TYPE 5 - ATOMIC PACKET		
23	ATOMIC test-and-swap transaction with data payload other than 1/2/4 bytes	Part 1, Section 4.1.7

S.No	Check Description	Specification Reference
24	ATOMIC swap transaction with data payload other than 1/ 2/4 bytes	Part 1, Section 4.1.7
25	Received Atomic compare and swap with data payload not equal to 16 bytes	Part 1, Section 4.1.7
TYPE 6 - SWRITE		
26	Data payload exceeds the maximum size of 256 bytes	Part 1, Section 4.1.8
27	Packet received without data payload	Part 1, Section 4.1.8
28	Data payload not multiple of Double word	Part 1, Section 4.1.8
29	Data payload lesser than one double word	Part 1, Section 4.1.8
TYPE 7 - LFC		
30	In the multiple PDU transfer case (in the middle of PDU transfer), after sending XOFF(ARB), receiving new PDUs after the current PDU transfer is done. Packets having CRF=1 is exceptional.	Part 9, Section 2.2.1
31	In the multiple PDU transfer case (in the middle of PDU transfer), after sending XOFF(ARB), no RELEASE packet is received after the current PDU transfer is completed.	Part 9, Section 2.2.1
32	When a request is pipelined, received XON(ARB), for the pipelined request before the current transaction has not been completed.	Part 9, Section 2.2.2
33	Received pipelined REQUEST before XON(ARB) was sent to the first REQUEST	Part 9, Section 2.2.2
34	LFC packets received with priority < 3	Part 9, Section 2.2.4
35	LFC packets received with CRF = 0, even though CRF is supported	Part 9, Section 2.2.4
36	Started receiving packets from a destination ID with a particular priority (Packets with CRF=1 is exceptional), for which the ‘number of XOFF packets sent’ is lesser than the ‘number of XON packets sent’ before the orphaned timer expires.	Part 9, Section 2.4.2.3
37	Receiving LFC packet, when there was no packets sent other than Maintenance and flow control transaction request flows.	Part 9, Section 2.4.3

S.No	Check Description	Specification Reference
38	Receiving second REQUEST message, before sending XOFF(ARB) for the previous REQUEST and before the first REQUEST is timed out	Part 9, Section 2.4.7
39	Received RELEASE with different flow that of the context is allocated for	Part 9, Section 2.4.7
40	Receiving REQUEST for pipelining, with the same sequence number, when the first PDU transaction is on progress	Part 9, Section 2.4.7
41	After sending XOFF for a particular priority, receiving packets with the same (Packets with CRF=1 is exceptional) or lower priorities before the orphaned timer expires. (For devices not supporting flow arbitration)	Part 9, Section 3.2, Section A.2
42	Received a LFC packet with data payload	Part 9, Section 3.2
43	Received XON(ARB), for a particular sequence number, when there was no REQUEST sent for the same sequence number	Part 9, Section 3.3
44	Received XOFF(ARB) for a particular sequence number, when there was no REQUEST sent for the same sequence number	Part 9, Section 3.3
45	Received RELEASE for a sequence number, for which there was no REQUEST received earlier.	Part 9, Section 3.3
46	Started receiving PDUs before the REQUEST - XON(ARB) hand shake happens	Part 9, Section 3.3
47	Started receiving PDUs before the XON(ARB) is sent for that sequence number	Part 9, Section 3.3
48	Received XON without XOFF preceded by that	
49	Received LFC packet with reserved flowID	Part 9, Section 3.2
TYPE 8 - MAINTENANCE		
50	Received maintenance read request for the size other than word, double word or multiple double word up to 64 bytes	Part 1, Section 4.1.10
51	Received maintenance write request for the size other than word, double word or multiple double word up to 64 bytes	Part 1, Section4.1.10
52	Received maintenance write request with the data payload size greater than wrsize mentioned in the packet	Part 1, Section4.1.10
53	Received maintenance write with reserved combination of wrsize and wdptr	Part 1, Section4.1.10

S.No	Check Description	Specification Reference
54	Maintenance response with hop count not equal to 8'hFF	Part 3, Section 2.5
TYPE 9 - DATA STREAMING		
55	Received a segment where the data payload size is greater than the MTU.	Part 10, Section 3.2.4
56	Data Streaming length field configured greater than Max-PDU supported	Part 10, Section 3.2.4
57	Received a start or single segment on an open context	Part 10, Section 3.2.5
58	Receiving End segment on a closed context	Part 10, Section 3.2.5
59	Receiving a continuation segment on a new context	Part 10, Section 3.2.5
60	The length of a reassembled PDU is lesser than the PDU length field in the end data streaming segment packet header	Part 10, Section 3.2.5
61	The length of a reassembled PDU is greater than the PDU length field in the end data streaming segment packet header	Part 10, Section 3.2.5
62	Received a segment where the data payload size is smaller than the MTU (End segment and single segment are exceptional)	Part 10, Section 3.2.5
63	Receiving more number of segments than Segsupport specified in 'Data Streaming Information CAR'	Part 10, Section 4.2
64	Received a packet with Odd bit set, when the data payload has an even number of half-words	Part 10, Section 4.2
65	Received a packet with Pad bit set, when the length of pay-load is multiple of 2 bytes	Part 10, Section 4.2
TYPE 9 - DS TRAFFIC MANAGEMENT		
66	Packet received with xh = 1 and any of S, E, O or P bits with non-zero values	Part 10, Section 4.3
67	After sending XOFF (DS traffic management) packet for a particular priority receiving DS packets with the same or lower priorities	Part 10, Section 4.3.2
68	After sending XOFF (DS traffic management) packet, receiving DS packets from the same destination, class and streamid.	Part 10, Section 4.3
69	Received XON (DS traffic management) without XOFF (DS traffic management) preceded by that	

S.No	Check Description	Specification Reference
70	Received packet with reserved xtype value	Part 10, Section 4.3
71	Received packet with reserved TM OP value	Part 10, Section 4.3
72	Received packet with valid TMOP and reserved parameter1 value	Part 10, Section 4.3
TYPE 10 - DOOR BELL		
73	Received Doorbell request packet with data payload	Part 2, Section 4.2.4
TYPE 11 - DATA MESSAGE		
74	Received a message segment with payload greater than the one specified in the ssize field	Part 2, Section 4.2.5
75	Received a message segment (except the last segment) with payload lesser than the one specified in the ssize field	Part 2, Section 4.2.5
76	Received duplicated/repeated message segment without any ‘retry’ response	Part 2, Section 4.2.5
77	Received message packet with reserved ssize	Part 2, Section 4.2.5
78	Received a message segment whose msgseg field is greater than msglen field	Part 2, Section 4.2.5
79	Packet received with Data payload lesser than 8 bytes (Minimum size of data payload in a message segment is 8 bytes)	Part 2, Section 4.2.5
80	Received a message segment without data payload	Part 2, Section 4.2.5
81	Received a message packet without double-word boundary alignment	Part 2, Section 4.2.5
82	Received message segments of the same message (except the last segment) with different payload size.	Part 2, Section 4.2.5
83	Received message segments of the same message with different ssize field.	Part 2, Section 4.2.5
84	Received message segments of the same message with different msg_length field.	Part 2, Section 4.2.5
85	Data Message Operation not completed at the end of simulation	NA
TYPE 13 - RESPONSE CLASS		
86	Received response to NREAD where the requested length and received data payload mismatches	Part 1, Section 3.3.1

S.No	Check Description	Specification Reference
87	Received response to an atomic transaction with data payload size not equal to 8 bytes	Part 1, Section 4.1.5
88	Received data payload in the response with ERROR status. (Maintenance Read Response is exceptional)	Part 1, Section 4.2.3
89	Received data payload in the response which does not require data payload	Part 1, Section 4.2.3
90	Received response to Doorbell packet with data payload	Part 2, Section 4.3.3
91	Received 'retry' response when expected for a 'error' response	

6.2 Transport Layer Checks

The transport layer monitor checks are listed in Table 17, "Transport Layer Checks," on page 91.

Table 17: Transport Layer Checks

S.No	Check Description	Specification reference
1	Received packet without interchanged source and destination address in the response packet as that of request packet.	Part 3. Section 2.3
2	Received a packet that contained a destination ID that is not defined for this processing element	Part 3. Section 2.3
3	Packets received with reserved 'tt' field.	Part 3. Section 2.4

6.3 PL Monitor checks

The physical layer monitor checks are listed in Table 18, "Physical Layer Checks," on page 91.

Table 18: Physical Layer Checks

S.No	Check Description	Specification reference
LINK PROTOCOL CHECKS		
1	Unexpected packet-accepted control symbol transmitted.	5.13.2.3.1

S.No	Check Description	Specification reference
2	Unexpected packet-accepted control symbol received.	5.13.2.3.1
3	Packet -accepted with unexpected ackID value transmitted.	5.13.2.3.1
4	Packet-accepted with unexpected ackID value received.	5.13.2.3.1
5	Device is operating in receiver-controlled flow control mode, and the buf status field is not equal to 0x1F in the transmitted packet-accepted control symbol.	5.9.1
6	Device is operating in receiver-controlled flow control mode, and the buf status field is not equal to 0x1F in the received packet-accepted control symbol.	5.9.1
7	Unexpected packet-retry control symbol transmitted.	5.13.2.3.1
8	Unexpected packet-retry control symbol received.	5.13.2.3.1
9	Packet -retry with unexpected ackID value transmitted.	5.13.2.3.1
10	Packet-retry with unexpected ackID value received.	5.13.2.3.1
11	Device is operating in receiver-controlled flow control mode, and the buf status field is not equal to 0x1F in the transmitted packet-retry control symbol.	5.9.1
12	Device is operating in receiver-controlled flow control mode, and the buf status field is not equal to 0x1F in the received packet-retry control symbol.	5.9.1
13	Unexpected packet-not-accepted control symbol transmitted.	5.13.2.3.1
14	Unexpected packet-not-accepted control symbol received.	5.13.2.3.1
15	Packet-not-accepted with invalid cause field transmitted.	3.4.3 / Table 3-4
16	Packet-not-accepted with invalid cause field received.	3.4.3 / Table 3-4
17	Status control symbol with unexpected ackID-status transmitted.	5.13.2.3.1
18	Status control symbol with unexpected ackID-status received.	5.13.2.3.1
19	Device is operating in receiver-controlled flow control mode, and the buf status field is not equal to 0x1F in the transmitted status control symbol.	5.9.1
20	Device is operating in receiver-controlled flow control mode, and the buf status field is not equal to 0x1F in the transmitted status control symbol.	5.9.1
21	Atleast one status control symbol have to be transmitted every 1024 / code-groups.	5.5.3.1

S.No	Check Description	Specification reference
22	Atleast one status control symbol have to be received every 1024 code-groups.	5.5.3.1
23	VC_status control symbol is transmitted when multiple VCs are not supported.	3.4.5
24	VC_status control symbol is received when multiple VCs are not supported.	3.4.5
25	Device is operating in receiver-controlled flow control mode, and the buf status field is not equal to 0x1F in the transmitted VC_status control symbol.	5.9.1
26	Device is operating in receiver-controlled flow control mode, and the buf status field is not equal to 0x1F in the transmitted VC_status control symbol.	5.9.1
27	Link-response control symbol with unexpected ackID-status transmitted.	5.13.2.3.1
28	Link-response control symbol with unexpected ackID-status received.	5.13.2.3.1
29	Reserved cause field value seen on the transmitted packet-not-accepted control symbol	-NA-
30	Reserved port-status value seen on the transmitted link-response control symbol.	-NA-
31	Reserved cause field value seen on the received packet-not-accepted control symbol	-NA-
32	Reserved port-status value seen on the received link-response control symbol.	-NA-
33	Unexpected link-response control symbol received when no link-request is outstanding.	5.7
34	Unexpected link-response control symbol transmitted when no link-request is outstanding.	5.7
35	SOP control symbol transmitted with /SC/ delimiter.	3.5 / Table 3-7
36	Packet transmission before link initialization is complete.	5.5.3.1
37	EOP control symbol transmitted with /SC/ delimiter.	3.5 / Table 3-7
38	EOP control symbol transmitted when a packet is not in progress.	3.5.3
39	STOMP control symbol transmitted when a packet is not in progress.	3.5 / Table 3-7
40	STOMP control symbol received when a packet is not in progress.	3.5 / Table 3-7

S.No	Check Description	Specification reference
41	Link-request control symbol received with PD delimiter when no packet is in progress.	3.5.5 / Table 3-7
42	Restart-from-retry control symbol received with PD delimiter when no packet is in progress.	3.5.5 / Table 3-7
43	Restart-from-retry control symbol transmitted without receiving a packet-retry control symbol.	3.5.4
44	Unexpected Restart-from-retry control symbol transmitted when there are no outstanding packets.	3.5.4
45	A new link request is transmitted when a previous link request is outstanding.	5.7
46	Sync sequence doesn't precede the link request control symbol.	4.8.2
47	Link request doesn't follow immediately after a sync sequence.	4.8.2
48	Invalid / Incomplete sync sequence transmitted.	4.8.2
49	Invalid / Incomplete sync sequence received.	4.8.2
50	Clock compensation sequence not transmitted in the last 5000 characters.	4.7.1
51	Packet retry control symbol transmitted when multiple VC is supported and enabled.	5.4 / 5.8
52	Packet retry control symbol received when multiple VC is supported and enabled.	5.4 / 5.8
53	Packet retry control symbol transmitted for a packet, the VC of which is operating in CT mode.	5.4 / 5.8
54	Packet retry control symbol received for a packet, the VC of which is operating in CT mode.	5.4 / 5.8
55	Input error detected, recovery not initiated. i.e., packet-not-accepted is not transmitted.	5.13.2.6
56	Received a packet-not-accepted control symbol, link request with input-status command is not sent out.	5.13.2.7
57	Received a link-request with input-status command, link response is not sent out.	5.7
58	Start flag is not set in first control symbol of timestamp sequence.	6.5.3.5
59	Start flag is set in other than first control symbol of the timestamp sequence.	6.5.3.5
60	End flag is set in other than last control symbol of the timestamp sequence	6.5.3.5

S.No	Check Description	Specification reference
61	End flag is not set in the lst control symbol of the times-tamp sequence.	6.5.3.5
62	Incomplete timestamp sequence encountered. Times-tamp sequnce shall not be truncated by any other control symbols.	6.5.3.5
IDLE SEQUENCE CHECKS		
63	Invalid / Illegal characters being transmitted in the IDLE sequence on lane <x>.	5.13.2.2
64	Invalid / Illegal characters received in the IDLE sequence on lane <x>.	5.13.2.2
65	IDLE1 sequence doesn't begin with /K/ character.	4.7.2
66	IDLE2 sequence neither begins with clock compensation sequence nor with D0.0	4.7.4
67	IDLE sequence is not identical across lanes.	5.13.2.2
68	Column of ||A|| not seen on the transmitted IDLE sequence.	5.13.2.2
69	Column of ||A|| not seen on the received IDLE sequence.	5.13.2.2
70	Column of ||K|| not seen on the transmitted IDLE sequence.	5.13.2.2
71	Column of ||K|| not seen on the received IDLE sequence.	5.13.2.2
72	Column of ||R|| not seen on the transmitted IDLE sequence.	5.13.2.2
73	Column of ||R|| not seen on the received IDLE sequence.	5.13.2.2
74	Column of ||M|| not seen on the transmitted IDLE sequence.	5.13.2.2
75	Column of ||M|| not seen on the received IDLE sequence.	5.13.2.2
76	Column of ||D0.0| not seen on the transmitted IDLE sequence.	5.13.2.2
77	Column of ||D0.0|| not seen on the received IDLE sequence.	5.13.2.2
78	In IDLE1 sequence, successive /A/ characters has to be separated by atleast 16 non-A characters.	4.7.2
79	The length of the pseudo random data field in the IDLE2 sequence is less than 509 characters.	4.7.4.1.1
80	The length of the pseudo random data field in the IDLE2 sequence is greater than 515 characters.	4.7.4.1.1

S.No	Check Description	Specification reference
81	The special characters within the pseudo-random data characters of the IDLE2 sequence have to be separated by at least 16 and not more than 31 D0.0 characters except the first contiguous sequence of pseudo-random data characters.	4.7.4.1.1
82	The length of first contiguous sequence of pseudo-random characters in the IDLE2 sequence shall be no less than 16 and no more 35 characters.	4.7.4.1.1
83	When the IDLE2 sequence is truncated at the pseudo-random data characters, the length of the last contiguous sequence of pseudo-random characters shall be no less than 4 and no more than 35 characters.	4.7.4.1.1
84	After an M character, 4 D0.0 characters has to follow and the sequence shall not be truncated before 4 D0.0 characters after an M character.	4.7.4
85	IDLE2 sequence shall not be truncated within the 4 M characters of the CS marker field.	4.7.4
86	5th and 7th characters of IDLE2 CS marker field has to be same.	4.7.4.1.2
87	6th and 8th characters of IDLE2 CS marker field has to be complement to each other.	4.7.4.1.2
88	Dx.y character of the IDLE2 CS marker indicates wrong lane number on lane <x>	4.7.4.1.2 / Table 4-6
89	Dx.y character of the IDLE2 CS marker indicates wrong active link width on lane <x>	4.7.4.1.2 / Table 4-5
90	The bits [0-31] and [32-63] of IDLE2 CS field has to bit-wise compliment to each other.	4.7.4.1.3
91	Port doesn't support IDLE2 sequence, but IDLE2 sequence is being transmitted on the link.	6.6.8
CONTROL SYMBOL CHECKS		
92	Invalid / Illegal character seen in the transmitted control symbol.	5.13.2.3.2
93	Invalid / Illegal character seen in the received control symbol.	5.13.2.3.2
94	Starting /SC/ delimiter occurs on lane, the modulo 4 of which is non-zero.	5.13.2.3.2
95	Wrong CRC in transmitted control symbol.	5.13.2.3.2
96	Wrong CRC in received control symbol.	5.13.2.3.2

S.No	Check Description	Specification reference
97	End de-limiter is missing at the 7th character position from the start de-limiter.	5.13.2.3.2
98	Start delimiter and end delimiter are not same.	5.13.2.3.2
PACKET CHECKS		
99	Invalid / Illegal character seen in the transmitted packet.	5.13.2.4
100	Invalid / Illegal character seen in the received packet.	5.13.2.4
101	Wrong CRC in transmitted packet.	5.13.2.4
102	Wrong CRC in received packet.	5.13.2.4
103	Packet transmitted with wrong ackID.	5.13.2.4
104	Packet received with wrong ackID.	5.13.2.4
105	Request packet transmitted with illegal priority 2'b11.	5.6.3
106	Request packet received with illegal priority 2'b11.	5.6.3
107	Packet is being transmitted when all the possible ackIDs are outstanding.	5.6.2
108	Packet is being received when all the possible ackIDs are outstanding.	5.6.2
109	Starting /PD/ delimiter of a packet occurs on a lane, the modulo 4 of which is non-zero.	5.13.2.3.2
110	Total packet size exceeds maximum supported bytes.	5.13.2.4
111	Packet is transmitted in output-error-stopped state.	5.13.2.7
TIMEOUT CHECKS		
112	Link timeout occured. Packet acknowledgement is not received for with ackID <x>	6.6.2
113	Link timeout occured. Link response is not received for an outstanding link request.	6.6.2
114	Link timeout occured. Loop-timing response is not received for an outstanding loop-timing request.	6.6.2
AET CHECKS		
115	Transmit equalizer command is transmitted when transmit equalizer control is not supported.	6.6.10 / Table 6-18
116	Transmit equalizer command is de-asserted before an ACK/NACK is received and also before 250us (100us incase of BRC3) from the time the command is asserted.	4.7.4.1.4
117	Transmit equalizer command ACK/NACK has to be de-asserted within 250us (100us incase of BRC3) from the time the equalizer command is de-asserted.	4.7.4.1.4

S.No	Check Description	Specification reference
118	Multiple equalization commands are set in the same IDLE2 CS field.	4.7.4.1.4
119	ACK and NACK are set together in the same IDLE2 CS field.	4.7.4.1.4
120	ACK / NACK is set when no AET command is outstanding.	4.7.4.1.4
121	AET command asserted again before the re-assertion timer expires.	4.7.4.1.4
122	AET command shall not change / de-asserted before an ACK / NACK is received or before the command time-out occurs.	4.7.4.1.4
123	AET command not de-asserted even after the ACK time-out occured.	4.7.4.1.4
GEN3 SPECIFIC CHECKS*		
124	Incorrect sequence number detected in the first control symbol of timestamp sequence.	6.5.3.5 / Table 6-2.
125	Incorrect sequence number detected in the second control symbol of timestamp sequence.	6.5.3.5 / Table 6-2.
126	Incorrect sequence number detected in the third control symbol of timestamp sequence.	6.5.3.5 / Table 6-2.
127	Incorrect sequence number detected in the fourth control symbol of timestamp sequence.	6.5.3.5 / Table 6-2.
128	Unexpected timestamp sequence control symbols detected.	6.5.3.5
129	Unexpected loop-request control symbol detected.	6.5.3.5
130	Unexpected loop-timing response control symbol is transmitted when no loop-timing request is outstanding.	6.5.3.5
131	Unexpected loop-timing response control symbol is received when no loop-timing request is outstanding.	6.5.3.5
132	Incomplete timestamp sequence control symbol encountered. Timestamp sequence control symbols shall not be interrupted.	6.5.3.5.1
133	SOP-padded control symbol transmitted, when no packet was previously transmitted.	3.5.1
134	Previous packet was not padded to achieve a total length that is multiple of 8 bytes, but it is terminated with SOP-padded control symbol.	3.5.1

S.No	Check Description	Specification reference
135	Previous packet was padded to achieve a total length that is multiple of 8 bytes, but it is terminated with SOP-unpadded control symbol.	3.5.1
136	Packet which is not padding appended to achieve a total length that is multiple of 8 bytes is terminated by EOP-padded control symbol.	3.5.3
137	Packet which is padding appended to achieve a total length that is multiple of 8 bytes is terminated by EOP-unpadded control symbol.	3.5.3
138	type and !type of a 67B codeword have to be compliment to each other.	5.5.1
139	Invalid skip-marker control codeword is transmitted. Fixed value is different.	5.5.3.1
140	Invalid skip-marker control codeword is received. Fixed value is different.	5.5.3.1
141	Skip-marker control codeword have to be transmitted only as part of skip ordered sequence.	5.5.3.1
142	Invalid lane_check control codeword is transmitted. Fixed value is different.	5.5.3.2 / Table 5-2
143	Invalid lane_check control codeword is received. Fixed value is different.	5.5.3.2 / Table 5-2
144	BIP-23 and iBIP-23 fields of lane_check control codeword are not compliment to each other.	5.5.3.2 / Table 5-2
145	Lane_check control codeword have to be transmitted only as part of skip ordered sequence.	5.5.3.2
146	Invalid skip control codeword is transmitted. Fixed value is different.	5.5.3.4
147	Invalid skip control codeword is received. Fixed value is different.	5.5.3.4
148	Incorrect Status / Control control codeword is transmitted. Field <x> is not matching the expected value.	5.5.3.5
149	Field <x> which belongs to the port scope of a status / control codeword should have same value on all operating lanes.	5.5.3.5
150	Field <x> which belongs to the Asym. port scope of a status / control control codeword should have same value on all operating lanes.	5.5.3.5
151	Ordered sequences shall not be truncated.	5.9.1

S.No	Check Description	Specification reference
152	Ordered sequences have to be identical across all the operating lanes except the lane specific fields.	5.8
153	A link request control symbol shall always be preceded by a seed ordered sequence.	5.8.1
154	Skip control codeword shall be transmitted only as a part of skip ordered sequence.	5.8.3
155	Lane-check control codeword shall be transmitted only as a part of skip ordered sequence.	5.8.3
156	Skip ordered sequence shall be transmitted atleast once for every 5000 codewords per lane.	5.8.3
157	Control symbols embedded in a packet shall always align to an 8-byte boundary relative to the beginning of the packet.	6.5.3.3
158	As part of IDLE3, the seed ordered sequence shall be transmitted atleast once for every 49 codewords transmitted per lane.	5.9.1
159	As part of IDLE3, the status / control ordered sequence shall be transmitted atleast once for every 18 to 49 codewords transmitted per lane.	5.9.1
160	The 2 consecutive status / control control codewords within a status / control ordered sequence have to be identical per lane.	5.8.2
161	Invalid / illegal codeword encountered inbetween IDLE sequence3.	5.9.1
162	CSB should be followed by either CSE or CSEB.	5.7
163	Incorrect Link CRC32 detected.	5.6
164	Transmit equalizer command shall not be asserted when status is other than not_updated.	5.10.2.2
165	Transmit equalizer command should be deasserted within 5us after 100us timeout.	5.10.2.2
166	Transmit equalizer command field shall not change before transmit equaizer status is received.	5.10.2.2
167	Transmit equalizer tap field shall not change before transmit equaizer status is received.	5.10.2.2
168	Transmit equalizer command should be deasserted within 5us after receiving status.	5.10.2.2

S.No	Check Description	Specification reference
169	Transmit equalizer status should not be other than not_updated when there is no active command-status handshake is outstanding.	5.10.2.2
170	Expected and actual coefficient status doesn't match.	802.3-2008, 72.6.10.2.4
171	Transmit width request is being transmitted when asymmetry mode is not supported.	7.6.14
172	Transmit width request is de-asserted before the transmit width request pending bit is asserted and also before 250us from the time the transmit width request is asserted.	5.16.1.3
173	Transmit width request pending bit has to be de-asserted within 250us from the time the transmit width request is de-asserted.	5.16.1.3
174	Receive width link command is being transmitted when asymmetry mode is not supported.	7.6.14
175	Receive width link command is de-asserted before the receive width command ACK/NACK is asserted and also before 62.5us from the time the receive width link command is asserted.	5.16.3
176	Receive width command ACK/NACK has to be de-asserted within 62.5us from the time the receive width link command is de-asserted.	5.16.3
177	Receive width link command ACK and Receive width link command NACK are asserted at the same time.	5.16.3
178	Receive width link command NACK is expected as unsupported / illegal link width requested, but receive width link command ACK is transmitted.	5.16.3

* Specification reference for LL checks, TL checks and PL GEN3 Specific Checks are wrt to Draft version 3.0.10. PL checks other than GEN3 Specific Checks are wrt Rev.2.2.

7 Call Backs

7.1 Logical Layer Callbacks

The following table shows the callbacks defined in the Logical Layer Agent.

Table 19: Logical Layer Callbacks

S.No	Call Back Name	Description
1	srio_ll_trans_generated	Called before sending a transaction to the transport layer agent. Object handle of srio_trans class is passed as an argument.
2	srio_ll_trans_received	Called when a transaction is received from the transport layer agent. Object handle of srio_trans class is passed as an argument.

Note: LL Callback usage can be referred in the file srio-vip/tests/tests/srio_ll_test_cb.sv

7.2 Transport Layer Callbacks

The following table shows the callbacks defined in the Transport Layer Agent.

Table 20: Transport Layer Callbacks

S.No	Call Back Name	Description
1	srio_tl_trans_generated	Called before sending a transaction to the physical layer agent. Object handle of srio_trans class is passed as an argument.
2	srio_tl_trans_received	Called when a transaction is received from the physical layer agent. Object handle of srio_trans class is passed as an argument.

Note: TL Callback usage can be referred in the file srio-vip/tests/tests/srio_tl_test_cb.sv

7.3 Physical Layer Callbacks

The following table shows the callbacks defined in the Physical Layer Agent.

Table 21: Physical Layer Callbacks

S.No	Call Back Name	Description
1	srio_pl_cg_generatedN *N- 0 to 15	Called before transmitting a code group to the SRIO Link.Object handle of srio_pl_cg class is passed as an argument.
2	srio_pl_cg_receivedN *N- 0 to 15	Called when a code group is received from the SRIO Link. Object handle of srio_pl_cg class is passed as an argument.
3	srio_pl_trans_generated	Called before transmitting a transaction on the SRIO link. Object handle of srio_trans class is passed as an argument.
4	srio_pl_trans_received	Called when a transaction is received on SRIO link. Object handle of srio_trans class is passed as an argument.
5	srio_pl_aet_cs_receivedN *N- 0 to 15	Called when a complete AET CS value is received.
6	srio_pl_trans_transmit	Called just before a Control Symbol or packet is stripped onto lanes. Object handle of srio_pl_data_trans and srio_trans is passed as argument
7	srio_pl_char_transmitted_laneN *N- 0 to 15	Called before each character is transmitted on lane

Note: PL Callback usage can be referred in the file srio-vip/tests/tests/srio_pl_test_cb.sv

7.4 Callbacks extended from uvm_report_catcher

The following table shows the callbacks to modify the severity of uvm reports.

Table 22: Report Catcher Callbacks

S.No	Call Back Name	Description
1	severity_modifier	Called when the uvm report is generated and modifies the severity of the uvm report with the given severity.
2	err_demoter	Called when the uvm ERROR report is generated and demotes the severity to WARNING.