SRIO VIP Quick Start Guide

V1.3

Revision History

Revision	Date	By	Change
1.0	04/26/2013	MV	Initial Version
1.1	06/03/2013	MV	Error related variables included
1.2	07/26/2013	MV	GEN3 Changes
1.3	09/20/2013	MV	GEN3 Changes and TXRX Model support

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

Contents

Figures

Tables

1 Introduction

1.1 Purpose

This document is a quick start guide which provides the information about setting up the verification environment using SRIO VIP. It also describes the configuration of the of SRIO VIP and its usage.

1.2 Scope

The document covers the environment setup, directory structure, configuration, integration and running the simulation.

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”*
- *“SRIO VIP Microarchitecture”*

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).

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

Figure 1, "Block Diagram Of SRIO VIP With DUT Setup," on page 10 shows the environment setup connecting the SRIO VIP with DUT. In tb_top module, interface and DUT are instantiated.Test is run from the tb_top module. In srio_base_test, srio_env is instantiated.

Figure 1 : Block Diagram Of SRIO VIP With DUT Setup

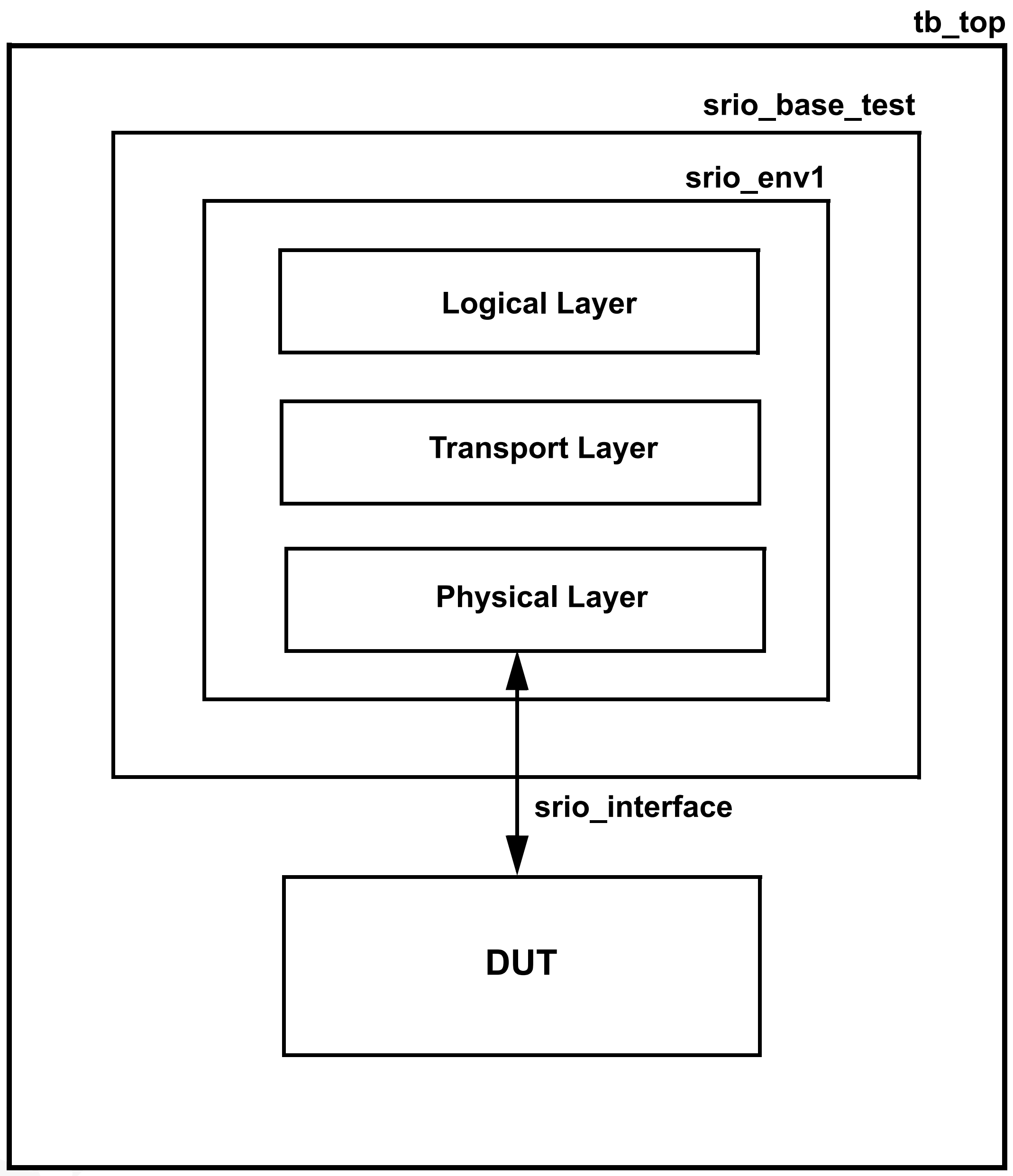
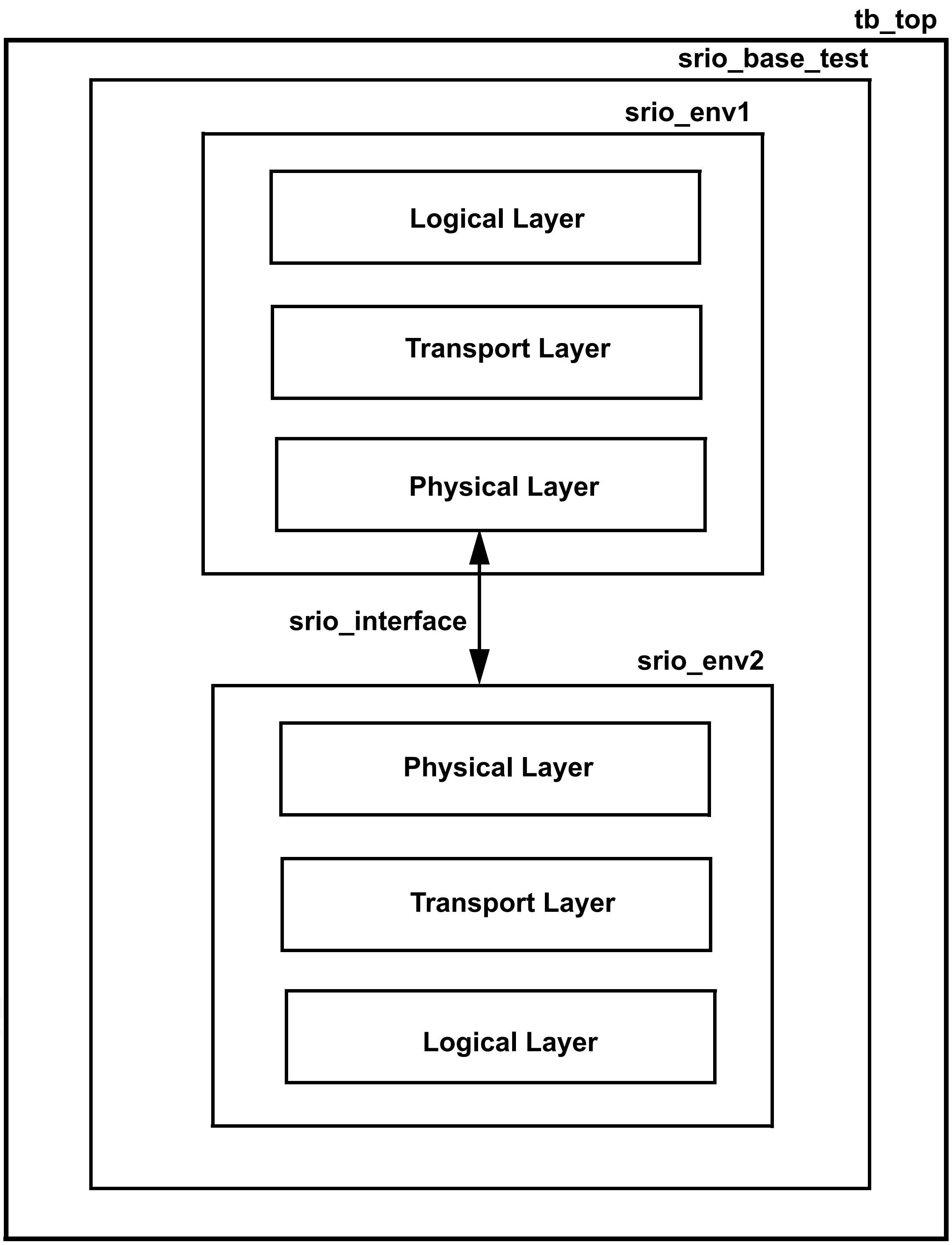


Figure 2, “Block Diagram Of SRIO VIP Back To Back Setup,” on page 11 shows the environment setup connecting the two SRIO VIPs back to back.In tb_top module, interface is instantiated two times one each for both SRIO VIPs. Test is run from the tb_top module. In srio_base_test, srio_env1 and srio_env2 are instantiated.

Figure 2 : Block Diagram Of SRIO VIP Back To Back Setup



3 Interface Description

All interface signals required for the SRIO VIP are declared in srio_interface.sv file.Refer to Table 1, “SRIO VIP Interface,” on page 12 for the signals description.

Table 1: SRIO VIP Interface

Name	Width	I/O	Function
srio_rst_n	logic	Input	Reset.
sim_clk	logic	Input	Simulation clock.Must be greater than serial clock divided by 10.Silence timer and discovery timer are executed based on posedge of sim_clk. Thus, inorder to match the DUT's timer value, the model's timer values have to programmed based on the sim_clk period. For example, if DUT's discovery timer is kept 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.
rx_sclk	logic[0:15]	Input	Serial RIO receive clock.
rxp	logic[0:15]	Input	Serial RIO receive data.
rxn	logic[0:15]	Input	Serial RIO receive complement data.
tx_sclk	logic[0:15]	Input	Serial RIO transmit clock.
txp	logic[0:15]	Output	Serial RIO transmit data.
txn	logic[0:15]	Output	Serial RIO transmit complement data.
rx_pclk	logic[0:15]	Input	Receive clock for parallel data mode.
rx_pdata	logic[0:15]	Input	Receive data for GEN1/GEN2 parallel data mode. Array of width 10-bits and depth 16.
tx_pclk	logic[0:15]	Input	Transmit clock for parallel data mode.
tx_pdata	logic[0:15]	Output	Transmit data for GEN1/GEN2 parallel data mode. Array of width 10-bits and depth 16.
gen3_rx_pdata	logic[0:15]	Input	Receive data for GEN3 parallel data mode. Array of width 67-bits and depth 16.
gen3_tx_pdata	logic[0:15]	Output	Transmit data for GEN3 parallel data mode. Array of width 67-bits and depth 16.

Name	Width	I/O	Function
TSG_clk	logic	Input	The TSG (Timestamp Generator) counter used in Timing Synchronization protocol runs on the TSG clock.

4 Configuration Variables

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 2, “SRIO VIP Global Configuration Parameters,” on page 14 for the list of global configuration variables and their description.

Table 2: 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.

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 3, "Logical Layer Configuration Parameters," on page 16 for the list of logical layer configuration variables and their description.

Table 3: 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.

S.No	Parameters	Description
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.
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.

S.No	Parameters	Description
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
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

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 4, "Transport Layer Configuration Parameters," on page 19 for list of transport layer configuration variables and their description.

Table 4: 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.

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 5, “PL Agent Configuration Parameters,” on page 19 for list of physical layer configuration variables and their description.

Table 5: 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

S.No	Parameter	Description
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
13	bfm_discovery_timer	Discovery timer value used by the BFM. It is executed based on the posedge of sim_clk. Thus, inorder 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.

S.No	Parameter	Description
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.
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.

S.No	Parameter	Description
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 interms 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
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.

S.No	Parameter	Description
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
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.

S.No	Parameter	Description
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 doesnot 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_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.
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, packets 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.

S.No	Parameter	Description
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 codeword lock state machine to count the valid codewords.
83	brc3_iv_slip	BRC3 IV_Slip value used in codeword 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 codewords.
85	lock_break_threshold	Number of invalid codewords to be received inorder 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.

S.No	Parameter	Description
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.
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.

S.No	Parameter	Description
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.
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.

S.No	Parameter	Description
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 }
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}

S.No	Parameter	Description
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}
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}

S.No	Parameter	Description
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
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

S.No	Parameter	Description
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
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

S.No	Parameter	Description
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
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

S.No	Parameter	Description
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
212	tminus3_prst_value	Codeword training Tapminus3 Preset Value
213	tminus3_impl_en	Codeword training Tapminus3 implementation enable

S.No	Parameter	Description
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.
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.

S.No	Parameter	Description
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 acknowleded. 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 acknowleded. Timeperiod is counted in terms of sim_clk period. Default value is 10000.

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 6, “PL User Input Parameters,” on page 36

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 6: 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 comand 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 comand 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.
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.

Table 6: PL User Input Parameters

S.No	Parameter	Description
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.

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. Table 7, "SRIO VIP's Sequence Item," on page 38 provides the description of various fields defined in this object.

Table 7: 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,DIS-COV-ERY,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,DISCOV-ERY,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,CSMARKER_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_err_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_response_en	bool	If TRUE, user directed response type is sent. If FALSE, response type is selected from LL configuration parameters
83	usr_directed_ll_reponse_type	srio_ll_resp_kind	LL_NO_RESP,LL_DONE,LL_ERROR,LL_RETRY
84	usr_directed_ll_response_delay	integer	Decides the latency of a LL response packet transmitted.
85	usr_directed_pl_response_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_response_delay	integer	Decides the latency of a PL response packet transmitted.
88	usr_directed_pl_ack_en	bool	If TRUE, user directed response type is sent. If FALSE, response type is selected from PL configuration parameters.
89	usr_directed_pl_ack_type	srio_pl_ack_kind	PL_ACCEPT,PL_NOT_ACCEPT,PL_RETRY.
90	usr_directed_pl_nac_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_a ck_delay	integer	Decides the latency of a 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_encountere d	bit	Set by LL monitor when any LL error is encountered
97	tl_err_encountere d	bit	Set by TL monitor when any TL error is encountered
98	pl_err_encountere d	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		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 Creating SV Environment Using SRIO VIP

This section explains how to use the SRIO VIP in a UVM based system verilog environment to build the DUT verification environment.

Creating tb_top module:

```
module tb_top;

// Import UVM and srio_test_lib package
import uvm_pkg::*;
import srio_test_lib_pkg::*;

// Instantiate the interfaces:
srio_interface SRIO_IF();

initial
begin
 uvm_config_db#(virtual  srio_interface)::set(null,  "*srio_env*",  "SRIO_VIF",  SRIO_IF);
 run_test();
end

//Assert Reset
initial
begin
   SRIO_IF.srio_rst_n = 0;
   #50ns;
   SRIO_IF.srio_rst_n = 1;
end

//Generate the clocks

// Instantiate DUT and connect the signals

end
endmodule
```

<u>PE MODEL:</u>

Creating base test:

```
`include "uvm_macros.svh"
import uvm_pkg::*;
import srio_env_pkg::*;

// Create base test from uvm_test
class srio_base_test extends uvm_test;

`uvm_component_utils(srio_base_test)
// Instantiate srio_env class
 srio_env env1;
```

```
srio_env_config env_config; // Global configuration object
srio_reg_block srio_reg_model_tx;  // Register Block for TX device, BFM
srio_reg_block srio_reg_model_rx;  // Register Block for RX Device, DUT

function void build_phase( uvm_phase phase );
//Create Global configuration object
env_config = srio_env_config::type_id::create("srio_env_config",this);
// Store the global config object handle to uvm config data base
uvm_config_db #(srio_env_config)::set(this,"*srio_env1*", "srio_env_config", env_config);

// Configure the global variables at this place
env_config.num_of_lanes = 4;
env_config.srio_mode    =  SRIO_GEN22;
env_config.srio_baud_rate = SRIO_5;

// Create Register blocks and build it
srio_reg_model_tx     = srio_reg_block::type_id::create("srio_reg_model_tx");
srio_reg_model_tx.build();
srio_reg_model_rx     = srio_reg_block::type_id::create("srio_reg_model_rx");
srio_reg_model_rx.build();

//Store the reg blocks handle to the global config object.Other components will take it from config
env_config.srio_reg_model_tx = srio_reg_model_tx;
env_config.srio_reg_model_rx = srio_reg_model_rx;

// SRIO Environment creation
env1 = srio_env::type_id::create("srio_env1", this );

//Configure the individual layer's configuration variable
env1.pl_agent.pl_agent_config.idle_sel  = 1;
env1.tl_agent.ll_config.has_checks       = 1;
env1.ll_agent.ll_config.interleaved_pkt  = FALSE;
endfunction
endclass
```

Creating LL test case which invokes nread sequence to create nread packets

```
class srio_ll_nread_req_test extends srio_base_test;

`uvm_component_utils(srio_ll_nread_req_test)

//LL nread virtual sequence which maps to actual nread sequence
srio_ll_nread_req_seq nread_req_seq;

task run_phase( uvm_phase phase );
nread_req_seq = srio_ll_nread_req_seq::type_id::create("nread_req_seq");
phase.raise_objection( this );
//Start the sequence using the virtual sequencer
nread_req_seq.start( env1.e_virtual_sequencer);
phase.drop_objection(this);
endtask
```

```
endclass
```

Creating srio_env:

```
class srio_env extends uvm_env;

`uvm_component_utils(srio_env);

srio_ll_agent ll_agent;  // LL Agent
srio_tl_agent tl_agent; // TL Agent
srio_pl_agent pl_agent; // PL Agent

srio_env_config env_config;
srio_virtual_sequencer e_virtual_sequencer; // Virtual Sequencer

// SRIO Reg Block
srio_reg_block srio_reg_model;
// Register layering adapter:
srio_reg_adapter srio_adapter;
// Register predictor:
uvm_reg_predictor #(srio_trans) srio_reg_predictor;
// SRIO Trans Item Decoder. Used by Register Model
srio_trans_decoder trans_decoder;

 // Functional coverage
 srio_ll_func_coverage srio_ll_fc;
 srio_tl_func_coverage srio_tl_fc;
 srio_pl_func_coverage srio_pl_fc;

 function void build_phase(uvm_phase phase);
// Get the environment config handle from uvm config data base
 if(!uvm_config_db #(srio_env_config)::get(this, "", "srio_env_config", env_config))
     `uvm_fatal("CONFIG FATAL", "Can't get the env_config")

// Storing the DUT reg block handle from config to local variable
srio_reg_model     = env_config.srio_reg_model_rx;
srio_adapter       = srio_reg_adapter::type_id::create("srio_adapter");
srio_reg_predictor = uvm_reg_predictor #(srio_trans)::type_id::create("srio_reg_predictor", this);
trans_decoder      = srio_trans_decoder::type_id::create("trans_decoder", this);

//Create LL,TL and PL Agents
ll_agent = srio_ll_agent::type_id::create("ll_agent", this);
tl_agent = srio_tl_agent::type_id::create("tl_agent", this);
pl_agent = srio_pl_agent::type_id::create("pl_agent", this);

// Create Virtual Sequencer
if(env_config.has_virtual_sequencer)
begin //{
e_virtual_sequencer = srio_virtual_sequencer::type_id::create("e_virtual_sequencer", this);
end //}
```

```
// Create Coverage Modules
if(env_config.has_coverage)
begin //{
if(env_config.srio_vip_model == SRIO_PE)
begin //{
srio_ll_fc     = srio_ll_func_coverage::type_id::create("srio_ll_fc", this);
srio_tl_fc     = srio_tl_func_coverage::type_id::create("srio_tl_fc", this);
end //}
srio_pl_fc     = srio_pl_func_coverage::type_id::create("srio_pl_fc", this);
end //}

endfunction

function void connect_phase(uvm_phase phase);
// Connect TLM Ports
ll_agent.ll_agent_tx_put_port.connect(tl_agent.tl_agent_tx_put_export);
tl_agent.tl_agent_tx_put_port.connect(pl_agent.pl_agent_tx_put_export);
pl_agent.pl_agent_rx_put_port.connect(tl_agent.tl_agent_rx_put_export);
tl_agent.tl_agent_rx_put_port.connect(ll_agent.ll_agent_rx_put_export);

// Analysis port to upper layer import connection
tl_agent.tl_monitor.tx_mon_ap.connect(ll_agent.ll_monitor.tx_monitor.ll_tx_mon_imp);
tl_agent.tl_monitor.rx_mon_ap.connect(ll_agent.ll_monitor.rx_monitor.ll_rx_mon_imp);
pl_agent.pl_monitor.tx_mon_ap.connect(tl_agent.tl_monitor.tx_monitor.tl_tx_mon_imp);
pl_agent.pl_monitor.rx_mon_ap.connect(tl_agent.tl_monitor.rx_monitor.tl_rx_mon_imp);

// Assigning LL,TL and PL sequencer handles to virtual sequencer handles
if(env_config.has_virtual_sequencer)
begin //{
e_virtual_sequencer.v_ll_sequencer = ll_agent.ll_sequencer;
e_virtual_sequencer.v_tl_sequencer = tl_agent.tl_sequencer;
e_virtual_sequencer.v_pl_sequencer = pl_agent.pl_sequencer;
end //}

// Register sequencer layering part:
if(env_config.srio_vip_model == SRIO_PE)
srio_reg_model.srio_reg_block_map.set_sequencer(ll_agent.ll_sequencer, srio_adapter);
else
srio_reg_model.srio_reg_block_map.set_sequencer(pl_agent.pl_sequencer, srio_adapter);

//Register Layer
// Register prediction part:
// Set the predictor Adress map:
srio_reg_predictor.map = srio_reg_model.srio_reg_block_map;
// Set the predictor adapter:
srio_reg_predictor.adapter = srio_adapter;
// Disable the register models auto-prediction
srio_reg_model.srio_reg_block_map.set_auto_predict(0);
trans_decoder.tx_decoder.srio_reg_model = srio_reg_model;
// Connect the predictor to the bus agent monitor analysis port
if(env_config.srio_vip_model == SRIO_PE)
begin //{
```

```
ll_agent.tx_mon_ap.connect(trans_decoder.tx_decoder.analysis_export);
ll_agent.rx_mon_ap.connect(trans_decoder.rx_decoder.analysis_export);
//FC modules are collecting packets from monitor analysis ports
if(env_config.has_coverage)
begin //{
ll_agent.tx_mon_ap.connect(srio_ll_fc.tx_trans_collector.analysis_export);
ll_agent.rx_mon_ap.connect(srio_ll_fc.rx_trans_collector.analysis_export);
tl_agent.tx_mon_ap.connect(srio_tl_fc.tx_trans_collector.analysis_export);
tl_agent.rx_mon_ap.connect(srio_tl_fc.rx_trans_collector.analysis_export);
pl_agent.tx_mon_ap.connect(srio_pl_fc.tx_trans_collector.analysis_export);
pl_agent.rx_mon_ap.connect(srio_pl_fc.rx_trans_collector.analysis_export);
srio_ll_fc.ll_agent = ll_agent;
srio_pl_fc.pl_agent = pl_agent;
end //}
end //}
else
begin //{
pl_agent.tx_mon_ap.connect(trans_decoder.tx_decoder.analysis_export);
 pl_agent.rx_mon_ap.connect(trans_decoder.rx_decoder.analysis_export);
if(env_config.has_coverage)
begin //{
pl_agent.tx_mon_ap.connect(srio_pl_fc.tx_trans_collector.analysis_export);
pl_agent.rx_mon_ap.connect(srio_pl_fc.rx_trans_collector.analysis_export);
srio_pl_fc.pl_agent = pl_agent;
end //}
end //}

// Connect the srio transaction decoder to the register predictor's input port
trans_decoder.ap.connect(srio_reg_predictor.bus_in);
endfunction: connect_phase
endclass
```

Creating virtual sequencer:

```
class srio_virtual_sequencer extends  uvm_sequencer #(srio_trans);
`uvm_component_utils(srio_virtual_sequencer)

srio_ll_sequencer v_ll_sequencer;
srio_tl_sequencer v_tl_sequencer;
srio_pl_sequencer v_pl_sequencer;

endclass
```

Creating Virtual Sequence:

```
class srio_virtual_base_seq extends uvm_sequence#(srio_trans);

  `uvm_object_utils(srio_virtual_base_seq)

// Config and reg model handles
 srio_env_config env_config;
 srio_reg_block srio_reg_model;
```

```
///Virtual sequencer Handles for LL,TL and PL sequencers
 srio_ll_sequencer seq_ll_sequencer;
 srio_tl_sequencer seq_tl_sequencer;
 srio_pl_sequencer seq_pl_sequencer;
 srio_virtual_sequencer seq_virtual_seqr;

task body();

assert($cast(seq_virtual_seqr,m_sequencer))

if(!uvm_config_db #(srio_env_config)::get(m_sequencer, "", "srio_env_config", env_config))
 `uvm_fatal("Config Fatal", "Can't get the env_config")

seq_ll_sequencer= seq_virtual_seqr.v_ll_sequencer;
seq_tl_sequencer= seq_virtual_seqr.v_tl_sequencer;
seq_pl_sequencer= seq_virtual_seqr.v_pl_sequencer;

endtask
endclass : srio_virtual_base_seq
```

Creating LL Nread Virtual Sequence:

```
class srio_ll_nread_req_seq extends srio_virtual_base_seq;
`uvm_object_utils(srio_ll_nread_req_seq)

//This is the original LL sequence which creates nread packets. This virtual sequence maps to
// that sequence
srio_ll_request_class_seq ll_nread_req_seq ;

task body();
super.body();

repeat (5) begin //{
ll_nread_req_seq = srio_ll_request_class_seq::type_id::create("ll_nread_req_seq");
// User can control some of the nread fields from virtual sequence
ll_nread_req_seq.ftype_0 = 4'h2;
ll_nread_req_seq.ttype_0 = 4'h4;
ll_nread_req_seq.rdsize_0 = 4'h6;
ll_nread_req_seq.SrcTID_0 = $random;
ll_nread_req_seq.ext_address_0= $urandom;
ll_nread_req_seq.address_0 = $random;
ll_nread_req_seq.xamsbs_0 = $random;
ll_nread_req_seq.wdptr_0 = $random;

ll_nread_req_seq.start(seq_ll_sequencer);
end //}
endtask
endclass : srio_ll_nread_req_seq
```

Creating LL Base Sequence:

```
class srio_ll_base_seq extends uvm_sequence#(srio_trans);

`uvm_object_utils(srio_ll_base_seq)

srio_env_config env_config;
srio_reg_block srio_reg_model;

task pre_body();
super.pre_body();
if(!uvm_config_db #(srio_env_config)::get(m_sequencer, "", "srio_env_config", env_config))
`uvm_fatal("Config Fatal", "Can't get the env_config")
srio_reg_model = env_config.srio_reg_model_rx;
//Wait for the
wait (env_config.pl_mon_tx_link_initialized == 1);
wait (env_config.pl_mon_rx_link_initialized == 1);
endtask

endclass : srio_ll_base_seq
```

Creating LL request packet transmitting sequence:

```
class srio_ll_request_class_seq extends srio_ll_base_seq;
`uvm_object_utils(srio_ll_request_class_seq)

srio_trans srio_trans_item;

//These fields are controlled from virtual sequence
 logic [3:0] ftype_0;
 logic [3:0] ttype_0;
 logic [31:0] ext_address_0;
 logic [28:0] address_0;
 logic [1:0] xamsbs_0;
 logic wdptr_0;
 logic [3:0] wrsize_0;
 logic [3:0] rdsize_0;
 logic [7:0] SrcTID_0;

virtual task body();

srio_trans_item = srio_trans::type_id::create("srio_trans_item");

//Values of some of the fields are forced from virtual sequence. So need to disable the related
// constraints
srio_trans_item.Ftype.constraint_mode(0);
srio_trans_item.Ttype.constraint_mode(0);
srio_trans_item.ext_adress_xamsbs.constraint_mode(0);
srio_trans_item.rdsize_0.constraint_mode(0);
srio_trans_item.Wdptr.constraint_mode(0);

start_item(srio_trans_item);
```

```
assert(srio_trans_item.randomize() with {ftype ==ftype_0 ;ttype == ttype_0 ;rdsize ==
rdsize_0;SrcTID == SrcTID_0;ext_address == ext_address_0;address == address_0;xamsbs ==
xamsbs_0;wdptr == wdptr_0;});

//Prints the srio_trans sequence item fields
srio_trans_item.print();

finish_item(srio_trans_item);
endtask

endclass : srio_ll_request_class_seq
```

Creating TL Virtual Sequence:

```
class srio_tl_pkt_tt_seq extends srio_virtual_base_seq;
`uvm_object_utils(srio_tl_pkt_tt_seq)
srio_tl_pkt_tt_base_seq tl_pkt_tt_seq ;

task body();
super.body();
repeat(5) begin
tl_pkt_tt_seq = srio_tl_pkt_tt_base_seq::type_id::create("tl_pkt_tt_seq");
tl_pkt_tt_seq.start(vseq_tl_sequencer);
end
endtask

endclass : srio_tl_pkt_tt_seq
```

Creating TL Sequence:

```
class srio_tl_pkt_tt_base_seq extends srio_tl_base_seq; //{
`uvm_object_utils(srio_tl_pkt_tt_base_seq)
srio_trans srio_trans_item;

rand bit [3:0] ftype_0,ttype_0;

virtual task body();

srio_trans_item = srio_trans::type_id::create("srio_trans_item");
srio_trans_item.Ftype.constraint_mode(0);
srio_trans_item.Ttype.constraint_mode(0);
srio_trans_item.wrsize_0.constraint_mode(0);

ftype_0 = $urandom_range(32'd6,32'd5);
start_item(srio_trans_item);

assert(srio_trans_item.randomize() with {ftype ==ftype_0 ;ttype == 4'h4;wrsize ==4'hB ;wdptr
==1'b0;});
for(int i=0; i<8; i++) begin
srio_trans_item.payload.push_back(i);
 end
finish_item(srio_trans_item);
```

```
endtask
endclass : srio_tl_pkt_tt_base_seq
```

Creating PL Virtual Sequence:

```
class srio_pl_nwrite_swrite_req_seq extends srio_virtual_base_seq;
`uvm_object_utils(srio_pl_nwrite_swrite_req_seq)

srio_pl_nwrite_swrite_class_base_seq pl_nwrite_swrite_seq ;

task body();
super.body();
repeat(5) begin
pl_nwrite_swrite_seq=
srio_pl_nwrite_swrite_class_base_seq::type_id::create("pl_nwrite_swrite_seq");
// Connect to PL virtual sequencer
pl_nwrite_swrite_seq.start(vseq_pl_sequencer);
end
endtask

endclass : srio_pl_nwrite_swrite_req_seq
```

Creating PL Sequence:

```
class srio_pl_nwrite_swrite_class_base_seq extends srio_ll_base_seq;
`uvm_object_utils(srio_pl_nwrite_swrite_class_base_seq)

srio_trans srio_trans_item;
rand bit [3:0] ftype_0;
rand bit [3:0] ttype_0;

virtual task body();
srio_trans_item = srio_trans::type_id::create("srio_trans_item");
srio_trans_item.pkt_type = SRIO_PL_PACKET;
srio_trans_item.Ftype.constraint_mode(0);
srio_trans_item.Ttype.constraint_mode(0);
srio_trans_item.wrsize_0.constraint_mode(0);
ftype_0 = $urandom_range(32'd6,32'd5);

start_item(srio_trans_item);
assert(srio_trans_item.randomize() with {ftype ==ftype_0;ttype == 4'h4;wrsize ==4'hB ;wdptr
==1'b0;});
for(int i=0; i<8; i++) begin //{
srio_trans_item.payload.push_back(i);
end //}
finish_item(srio_trans_item);
endtask

endclass : srio_pl_nwrite_swrite_class_base_seq
```

PL Setup:

Creating base test:

```
`include "uvm_macros.svh"
import uvm_pkg::*;
import srio_env_pkg::*;

// Create base test from uvm_test
class srio_base_test extends uvm_test;

`uvm_component_utils(srio_base_test)
// Instantiate srio_env class
 srio_env env1;

srio_env_config env_config; // Global configuration object
srio_reg_block srio_reg_model_tx;  // Register Block for TX device, BFM
srio_reg_block srio_reg_model_rx;  // Register Block for RX Device, DUT

function void build_phase( uvm_phase phase );
//Create Global configuration object
env_config = srio_env_config::type_id::create("srio_env_config",this);
// Store the global config object handle to uvm config data base
uvm_config_db #(srio_env_config)::set(this,"*srio_env1*", "srio_env_config", env_config);

// Configure the global variables at this place
env_config.num_of_lanes = 4;
env_config.srio_mode    =  SRIO_GEN22;
env_config.srio_baud_rate = SRIO_5;

// Configure the model as PL model
 env_config1.srio_vip_model = SRIO_PL;

// Create Register blocks and build it
srio_reg_model_tx     = srio_reg_block::type_id::create("srio_reg_model_tx");
srio_reg_model_tx.build();
srio_reg_model_rx     = srio_reg_block::type_id::create("srio_reg_model_rx");
srio_reg_model_rx.build();

//Store the reg blocks handle to the global config object.Other components will take it from config
env_config.srio_reg_model_tx = srio_reg_model_tx;
env_config.srio_reg_model_rx = srio_reg_model_rx;

// SRIO Environment creation
env1 = srio_env::type_id::create("srio_env1", this );

//Configure PL Layer's configuration variable
env1.pl_agent.pl_agent_config.idle_sel  = 1;
endfunction
endclass
```

Creating srio_env:

```
class srio_env extends uvm_env;

`uvm_component_utils(srio_env);

srio_ll_agent ll_agent;  // LL Agent
srio_tl_agent tl_agent; // TL Agent
srio_pl_agent pl_agent; // PL Agent

srio_env_config env_config;
srio_virtual_sequencer e_virtual_sequencer; // Virtual Sequencer

// SRIO Reg Block
srio_reg_block srio_reg_model;
// Register layering adapter:
srio_reg_adapter srio_adapter;
// Register predictor:
uvm_reg_predictor #(srio_trans) srio_reg_predictor;
// SRIO Trans Item Decoder. Used by Register Model
srio_trans_decoder trans_decoder;

 // Functional coverage
 srio_ll_func_coverage srio_ll_fc;
 srio_tl_func_coverage srio_tl_fc;
 srio_pl_func_coverage srio_pl_fc;

// Users needs to connect the TLM and analysis port from PL agents to their TL/LL model.
uvm_put_port   #(srio_trans) tl_agent_tx_put_port; ///< Dummy TLM port used in PL model
uvm_put_export #(srio_trans) tl_agent_rx_put_export;  ///< Dummy TLM port used in PL model
uvm_tlm_fifo   #(srio_trans) tl_rx_fifo;              ///< Dummy TLM port fifo

 function void build_phase(uvm_phase phase);
// Get the environment config handle from uvm config data base
 if(!uvm_config_db #(srio_env_config)::get(this, "", "srio_env_config", env_config))
     `uvm_fatal("CONFIG FATAL", "Can't get the env_config")

// Storing the DUT reg block handle from config to local variable
srio_reg_model     = env_config.srio_reg_model_rx;
srio_adapter       = srio_reg_adapter::type_id::create("srio_adapter");
srio_reg_predictor = uvm_reg_predictor #(srio_trans)::type_id::create("srio_reg_predictor", this);
trans_decoder      = srio_trans_decoder::type_id::create("trans_decoder", this);

//Create LL,TL and PL Agents
ll_agent = srio_ll_agent::type_id::create("ll_agent", this);
tl_agent = srio_tl_agent::type_id::create("tl_agent", this);
pl_agent = srio_pl_agent::type_id::create("pl_agent", this);

// Create Virtual Sequencer
if(env_config.has_virtual_sequencer)
begin //{
e_virtual_sequencer = srio_virtual_sequencer::type_id::create("e_virtual_sequencer", this);
```

```
end //}

// Create Coverage Modules
if(env_config.has_coverage)
begin //{
if(env_config.srio_vip_model == SRIO_PE)
begin //{
srio_ll_fc     = srio_ll_func_coverage::type_id::create("srio_ll_fc", this);
srio_tl_fc     = srio_tl_func_coverage::type_id::create("srio_tl_fc", this);
end //}
srio_pl_fc     = srio_pl_func_coverage::type_id::create("srio_pl_fc", this);
end //}

if(env_config.srio_vip_model == SRIO_PL)
begin
 tl_agent_rx_put_export = new("tl_agent_rx_put_export", this);
 tl_agent_tx_put_port   = new("tl_agent_tx_put_port", this);
 tl_rx_fifo = new("tl_rx_fifo", this,100);
end

endfunction

function void connect_phase(uvm_phase phase);

if(env_config.srio_vip_model == SRIO_PE)  // If PE model,connect PL-<>TL and TL<-> LL
begin  // TLM ports
 ll_agent.ll_agent_tx_put_port.connect(tl_agent.tl_agent_tx_put_export);
 tl_agent.tl_agent_tx_put_port.connect(pl_agent.pl_agent_tx_put_export);
 pl_agent.pl_agent_rx_put_port.connect(tl_agent.tl_agent_rx_put_export);
 tl_agent.tl_agent_rx_put_port.connect(ll_agent.ll_agent_rx_put_export);
 // Analysis port to upper layer import connection
 tl_agent.tl_monitor.tx_mon_ap.connect(ll_agent.ll_monitor.tx_monitor.ll_tx_mon_imp);
 tl_agent.tl_monitor.rx_mon_ap.connect(ll_agent.ll_monitor.rx_monitor.ll_rx_mon_imp);
 pl_agent.pl_monitor.tx_mon_ap.connect(tl_agent.tl_monitor.tx_monitor.tl_tx_mon_imp);
 pl_agent.pl_monitor.rx_mon_ap.connect(tl_agent.tl_monitor.rx_monitor.tl_rx_mon_imp);
 end
 else if(env_config.srio_vip_model == SRIO_PL)
  begin
   // In PL model only PL agent will be included.User needs to connect the
   // PL agents's TLM and Analysis port to their wrapper logic.Here dummy
   // ports are connected for example and user needs to replace it.
    tl_agent_rx_put_export.connect(tl_rx_fifo.put_export);
    tl_agent_tx_put_port.connect(pl_agent.pl_agent_tx_put_export);
    pl_agent.pl_agent_rx_put_port.connect(tl_agent_rx_put_export);
end

// Assigning LL,TL and PL sequencer handles to virtual sequencer handles
if(env_config.has_virtual_sequencer)
begin //{
e_virtual_sequencer.v_ll_sequencer = ll_agent.ll_sequencer;
e_virtual_sequencer.v_tl_sequencer = tl_agent.tl_sequencer;
e_virtual_sequencer.v_pl_sequencer = pl_agent.pl_sequencer;
```

```
end //}

// Register sequencer layering part:
if(env_config.srio_vip_model == SRIO_PE)
srio_reg_model.srio_reg_block_map.set_sequencer(ll_agent.ll_sequencer, srio_adapter);
else
srio_reg_model.srio_reg_block_map.set_sequencer(pl_agent.pl_sequencer, srio_adapter);

//Register Layer
// Register prediction part:
// Set the predictor Adress map:
srio_reg_predictor.map = srio_reg_model.srio_reg_block_map;
// Set the predictor adapter:
srio_reg_predictor.adapter = srio_adapter;
// Disable the register models auto-prediction
srio_reg_model.srio_reg_block_map.set_auto_predict(0);
trans_decoder.tx_decoder.srio_reg_model = srio_reg_model;
// Connect the predictor to the bus agent monitor analysis port
if(env_config.srio_vip_model == SRIO_PE)
begin //{
ll_agent.tx_mon_ap.connect(trans_decoder.tx_decoder.analysis_export);
ll_agent.rx_mon_ap.connect(trans_decoder.rx_decoder.analysis_export);
//FC modules are collecting packets from monitor analysis ports
if(env_config.has_coverage)
begin //{
ll_agent.tx_mon_ap.connect(srio_ll_fc.tx_trans_collector.analysis_export);
ll_agent.rx_mon_ap.connect(srio_ll_fc.rx_trans_collector.analysis_export);
tl_agent.tx_mon_ap.connect(srio_tl_fc.tx_trans_collector.analysis_export);
tl_agent.rx_mon_ap.connect(srio_tl_fc.rx_trans_collector.analysis_export);
pl_agent.tx_mon_ap.connect(srio_pl_fc.tx_trans_collector.analysis_export);
pl_agent.rx_mon_ap.connect(srio_pl_fc.rx_trans_collector.analysis_export);
srio_ll_fc.ll_agent = ll_agent;
srio_pl_fc.pl_agent = pl_agent;
end //}
end //}
else
begin //{
pl_agent.tx_mon_ap.connect(trans_decoder.tx_decoder.analysis_export);
 pl_agent.rx_mon_ap.connect(trans_decoder.rx_decoder.analysis_export);
if(env_config.has_coverage)
begin //{
pl_agent.tx_mon_ap.connect(srio_pl_fc.tx_trans_collector.analysis_export);
pl_agent.rx_mon_ap.connect(srio_pl_fc.rx_trans_collector.analysis_export);
srio_pl_fc.pl_agent = pl_agent;
end //}
end //}

// Connect the srio transaction decoder to the register predictor's input port
trans_decoder.ap.connect(srio_reg_predictor.bus_in);
endfunction: connect_phase
endclass
```

TXRX Setup:

Creating base test:

```
// Create base test from uvm_test
class srio_base_test extends uvm_test;

// Configure the model as TXRX model
env_config1.srio_vip_model = SRIO_TXRX;

// Refer to PE/PL model's srio_base_test for remaining part of the code.

endclass
```

Creating srio_env:

```
class srio_env extends uvm_env;

// Refer to PE/PL model's env for remaining part of the code

// User needs to connect the PL agent's TLM to their wrapper logic.
uvm_put_port   #(srio_trans) tl_agent_tx_put_port;    ///< Dummy TLM port
uvm_put_export #(srio_trans) tl_agent_rx_put_export;  ///< Dummy TLM port
uvm_tlm_fifo   #(srio_trans) tl_rx_fifo;              ///< Dummy TLM port fifo

function void build_phase(uvm_phase phase);

if(env_config.srio_vip_model == SRIO_TXRX)
begin
tl_agent_rx_put_export = new("tl_agent_rx_put_export", this);
tl_agent_tx_put_port   = new("tl_agent_tx_put_port", this);
tl_rx_fifo = new("tl_rx_fifo", this,100);
end

endfunction

function void connect_phase(uvm_phase phase);

if(env_config.srio_vip_model == SRIO_TXRX)
begin
// In PL/TXRX model only PL agent will be included.User needs to connect the
// PL agents's TLM and Analysis port to their wrapper logic.Here dummy
// ports are connected for example and user needs to replace it.
tl_agent_rx_put_export.connect(tl_rx_fifo.put_export);
tl_agent_tx_put_port.connect(pl_agent.pl_agent_tx_put_export);
pl_agent.pl_agent_rx_put_port.connect(tl_agent_rx_put_export);
end

endfunction

endclass
```

Promoting/Demoting uvm report severity

A callback ‘severity_modifier’ extended from ‘uvm_report_catcher’ (uvm callback) is provided to promote/demote the uvm report’s severity.

The usage is as follows

At the beginning of the run_phase of the testcase, add the below lines.

```
/----------------------------------------------------------------------------------------------------------------/
severity_modifier   <hangle of  severity_modifier>    =    new;
<hangle of severity_modifier>.config_severity("<uvm report id>",    <Required severity>);
uvm_report_cb::add(null,    <hangle of severity_modifier>);
/----------------------------------------------------------------------------------------------------------------/
```

In the above code,

-> Line1: The handle of the callback (severity_modifier) is created and newed.

-> Line2: The task config_severity is called by passing the arguments, ‘uvm report id’ and ‘required severity’

-> Line3: The created callback is added for use

Example

The testcase srio_ll_no_payload_error_demote_test.sv shows the example of error demotion.

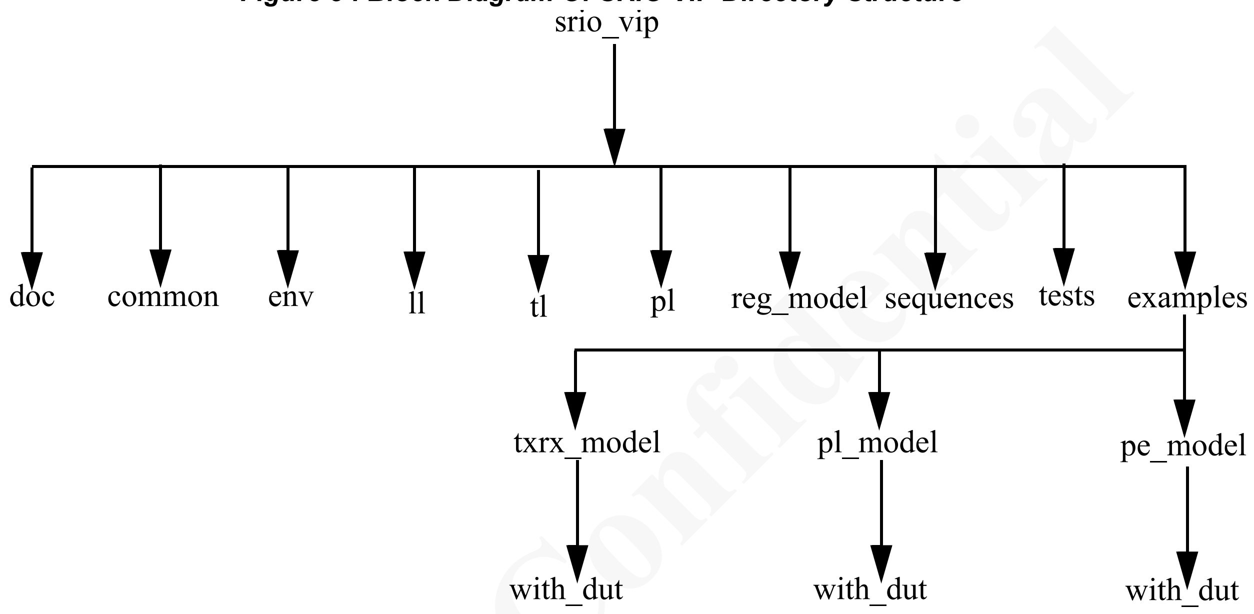
```
/----------------------------------------------------------------------------------------------------------------/
// Error Demotion

severity_modifier severity_modifier1 = new;
severity_modifier1.config_severity("SRIO_LL_PROTOCOL_CHECKER:NO_PAYLOAD_ERR",
UVM_WARNING);
uvm_report_cb::add(null, severity_modifier1);
/----------------------------------------------------------------------------------------------------------------/
```

The above examples describe about creating tb_top, tests,env and sequences.Same manner other sequences and test cases are created. User can also create their own test cases, sequences, env and build the required DUT verification setup.

7 Directory Structure

Figure 3, "Block Diagram Of SRIO VIP Directory Structure," on page 60 shows the SRIO VIP directory structure.

Figure 3 : Block Diagram Of SRIO VIP Directory Structure



7.1 Directory / Files Description

Table 8: Directory and Files Description

Name of the Directory/File	Description
doc	This directory contains the SRIO VIP documents.
common	This directory contains the files commonly used by SRIO VIP components.
srio_interface.sv	All the signals are declared here.
srio_base_trans.sv	Base class for srio_trans sequence item class and contains the sequence item variables and the constraints.
srio_trans.sv	Contains the methods of sequence item.
srio_report_catcher_callback.sv	Contains callbacks to promote/demote uvm report severity
env	This directory contains the environment related files.
srio_env_config.vh	Contains the environment configuration class and its variables.
srio_env.sv	srio_env class and its logic are included in this files

Name of the Directory/File	Description
srio_env_pkg.sv	Contains the srio_env_pkg package that includes all the agent, functional coverage and register model files.
srio_virtual_sequencer.sv	Contains the srio virtual sequencer class. This class has the handles of LL,TL and PL sequencers.
ll	All Logical Layer related files are included here.
srio_ll_agent.sv	Logical Layer agent's top level component. This class instantiates LL BFM(active), monitor(Passive) and configuration components.
srio_ll_variables.sv	Contains the Enum declarations used by the logical layer components.
srio_ll_config.vh	Logical Layer configuration class.
srio_ll_sequencer.sv	Logical Layer sequencer
srio_ll_bfm.sv	Logical Layer active BFM component.
srio_ll_base_generator.sv	LL base generator class.
srio_logical_transaction_generator.sv	This file contains the srio_logical_transaction_generator class. This class instantiates other packets generators and also includes the RR and WRR mechanism.
srio_io_generator.sv	Contains srio_io_generator class and its variables.
srio_msg_db_generator.sv	Contains srio_msg_db_generator class and handles generation of message and doorbell packets.
srio_ds_generator.sv	Contains srio_ds_generator class and handles generation of data streaming packets.
srio_gsm_generator.sv	Contains srio_gsm_generator class and its variables.
srio_lfc_generator.sv	Contains srio_lfc_generator class logic.
srio_resp_generator.sv	Contains srio_resp_generator class and handles generation of response packets for received requests.
srio_packet_handler.sv	Contains srio_packet_handler class and decodes the received packets.
srio_ll_monitor.sv	Top Level component of Logical Layer agent's Passive component. It instantiates srio_ll_txrx_monitor for TX and RX monitor checks.
srio_ll_txrx_monitor.sv	Contains srio_ll_txrx_monitor class and performs the protocol checks of Logical Layer. It instantiates LFC, MSG and DS assembly classes.
srio_ll_ds_assembly.sv	Contains srio_ll_ds_assembly class which is used for DS assembly and testing in LL monitor.
srio_ll_msg_assembly.sv	Contains srio_ll_msg_assembly class which is used for Data Message assembly and testing in LL monitor.

Name of the Directory/File	Description
srio_ll_lfc_assembly.sv	Contains srio_ll_lfc_assembly class which is used for LFC testing in LL monitor.
srio_ll_tx_trans_collector.sv	Instantiated inside srio_ll_func_coverage class. It receives transactions from ll agent tx monitor.
srio_ll_rx_trans_collector.sv	Instantiated inside srio_ll_func_coverage class. It receives transactions from ll agent rx monitor.
srio_ll_func_coverage.sv	Contains LL layer related coverpoints.
srio_ll_callback.sv	Contains the virtual task declarations used for logical layer callbacks.
tl	All Transport Layer related files are included here.
srio_tl_agent.sv	Transport Layer agent's top level component. This class instantiates TL BFM(active), monitor(Passive) and configuration components.
srio_tl_config.vh	Transport Layer configuration class.
srio_tl_sequencer.sv	TRansport Layer sequencer.
srio_tl_bfm.sv	Transport Layer active BFM component.
srio_tl_generator.sv	Contains srio_tl_generator class and handles generation of TL fields.
srio_tl_receiver.sv	Contains srio_tl_receiver class and decodes the received packets.
srio_tl_monitor.sv	Top Level component of TL agent's Passive component. It instantiates srio_tl_txrx_monitor for TX and RX monitor checks.
srio_tl_txrx_monitor.sv	Contains srio_tl_txrx_monitor class and performs the protocol checks of TL.
srio_tl_tx_trans_collector.sv	Instantiated inside srio_tl_func_coverage class. It receives transactions from tl agent tx monitor.
srio_tl_rx_trans_collector.sv	Instantiated inside srio_tl_func_coverage class. It receives transactions from tl agent rx monitor.
srio_tl_func_coverage.sv	Contains TL layer related coverpoints.
srio_tl_callback.sv	Contains the virtual task declarations used for transport layer callbacks.
pl	This directory contains the Physical Layer files.
srio_pl_agent.sv	Physical Layer agent's top level component. This class instantiates PL BFM(active), monitor(Passive) and configuration components.
srio_pl_variables.sv	Contains the Enum declarations used by the physical layer components.

Name of the Directory/File	Description
srio_pl_config.sv	Physical Layer configuration class.
srio_pl_sequencer.sv	Physical Layer sequencer.
srio_pl_driver.sv	Physical Layer Active BFM component.
srio_pl_pktcs_merger.sv	Physical layer packet and control symbol merger component.
srio_pl_idle_gen.sv	Physical layer idle generation and striping component.
srio_pl_data_trans.sv	Physical layer component that contains either the merged packet/control symbol or separate control symbol.
srio_pl_lane_data.sv	Carries the lane specific data such as 10bit codegroup, 8b data / control character etc.
srio_pl_lane_handler.sv	Receives the serial data and performs the decoding, descrambling, lane synchronization and receiver training. There will a separate instance of this class for each of the lanes supported.
srio_pl_rx_data_handler.sv	Processes each lane's data, detects and decodes the idle sequence, performs de-skewing and de-striping and passes the aligned data to state-machine class. It also forms the control symbol trans or packet trans whenever they are received and passes to the higher level component.
srio_pl_state_machine.sv	Contains the Align state machine and Initialization state machine.
srio_pl_pkt_handler.sv	Physical layer packet handler component.Contains logic for collecting the packets and control symbols.If a packet is detected, it is forwarded to upper layers and if control symbols detected it is sent to tx components for further processing.
srio_pl_monitor.sv	Top level class of PL agent's passive component. It instantiates the Tx monitor and Rx monitor w.r.to physical layer.
srio_pl_link_monitor.sv	PL agent's Tx monitor and Rx monitor.
srio_pl_protocol_checker.sv	Performs the protocol checks on the received control symbols and packets. It also performs link maintenance protocol checks.
srio_pl_tx_trans_collector.sv	Instantiated inside srio_pl_func_coverage class. It receives transactions from pl agent tx monitor.
srio_pl_rx_trans_collector.sv	Instantiated inside srio_pl_func_coverage class. It receives transactions from pl agent rx monitor.
srio_pl_func_coverage.sv	Contains PL layer related coverpoints.
srio_pl_fc_macro.sv	Contains macro definitions used in PL functional coverage

Name of the Directory/File	Description
srio_pl_callback.sv	Contains the virtual task declarations used for physical layer callbacks.
srio_gen_trans_tracker.sv	Records the transaction details of both env1 & env2 in two separate files inside examples/pe_model env1_srio_tracker.txt - Tx and Rx transactions of ENV1 env2_srio_tracker.txt - Tx and Rx transactions of ENV2
reg_model	This directory contains the register model files
srio_reg_block.sv	Contains all the srio memory mapped registers and the associated functional coverage bins for each register.
srio_reg_adapter.sv	This block converts SRIO transactions (Maintenance, NWRITE_R/NREAD) into register transactions using the bus2reg function.
srio_tx_trans_decoder.sv	This block decodes the MAINT_RD, MAINT_WR, NWRITE_R, NREAD transaction requests transmitted by BFM. This is a sub-block instantiated inside srio_trans_decoder.
srio_rx_trans_decoder.sv	This block decodes the responses received by BFM from DUT. This is a sub-block instantiated inside srio_trans_decoder.
srio_trans_decoder.sv	This passes the decoded request and response packets from srio_tx_trans_decoder and srio_rx_trans_decoder respectively to register predictor.
sequences	This directory contains all the sequences related files.
srio_ll_sequence_lib.sv	Contains the Logical Layer sequences.
srio_tl_sequence_lib.sv	Contains the Transport Layer sequences.
srio_pl_sequence_lib.sv	Contains the Physical Layer sequences.
srio_virtual_sequence_lib.sv	Contains the virtual sequences.
srio_seq_lib_pkg.sv	Sequences library package includes LL,TL,PL abd Virtual sequences.
tests	Contains the test case files.
srio_test_lib_pkg.sv	Test Cases library package include all test cases.
examples	This directory includes the examples files.
pe_model	Contains the scripts and tb_top for running simulation in PE model back to back setup.
srio_base_test.sv	Contains srio_base_test test case for PE model setup.
tb_top.sv	Top module for PE model back to back setup.
RUN	RUN scripts gets the PE model back to back simulation options and invokes make file.

Name of the Directory/File	Description
makefile.sim	make file to run the back to back setup simulation for PE model.
with_dut	This directory includes the example file for creating the SRIO VIP with DUT integrated setup for PE model.
tb_top.sv	Top module for DUT setup with PE model.
srio_base_test.sv	Base test case for DUT setup with PE model.
pl_model	Contains the scripts and tb_top for running simulation in PL model back to back setup.
srio_base_test.sv	Contains srio_base_test test case for PL model.
tb_top.sv	Top module for back to back setup.
RUN	RUN scripts gets the PL model back to back simulation options and invokes make file.
makefile.sim	make file to run the back to back setup simulation for PL model.
with_dut	This directory includes the example file for creating the SRIO VIP with DUT integrated setup for PL model.
tb_top.sv	Top module for DUT setup with PL model.
srio_base_test.sv	Base test case for DUT setup with PL model.
txrx_model	Contains the scripts and tb_top for running simulation in TXRX model back to back setup.
srio_base_test.sv	Contains srio_base_test test case for TXRX model.
tb_top.sv	Top module for back to back setup.
RUN	RUN scripts gets the TXRX model back to back simulation options and invokes make file.
makefile.sim	make file to run the back to back setup simulation for TXRX model.
with_dut	This directory includes the example file for creating the SRIO VIP with DUT integrated setup for TXRX model.
tb_top.sv	Top module for DUT setup with TXRX model.
srio_base_test.sv	Base test case for DUT setup with TXRX model.

8 Running Simulation in Demo Setup

This chapter explains about setting up the SRIO VIP back to back environment and running the simulation. For creating DUT integrated setup, user needs to create the top verilog module and srio base test as discussed in the previous sections. User also need to update the scripts accordingly.

Set the SRIO VIP Path.

setenv SRIO_VIP_PATH <Path of the srio-vip directory>

Set the UVM library path.

setenv UVM_PATH <Path of the UVM library>

VCS users need to also set the following variable

setenv VCS_UVM_HOME $UVM_PATH/src

PE Model

Run the simulation from srio-vip/examples/pe_model directory.

Example RUN command:

./RUN -test srio_ll_nread_req_test

PL Model

Run the simulation from srio-vip/examples/pl_model directory.

Example RUN command:

./RUN -test srio_pl_nwrite_swrite_req_test

TXRX Model

Run the simulation from srio-vip/examples/txrx_model directory.

Example RUN command:

./RUN -test srio_txrx_model_test

Table 9: RUN Command Options

Option	Description
-s or -sim	Simulator. One of nc,vcs and questa. Default is nc.
-t or -test	Name of the test. One of the tests from test library.
-l or -lane	Number of Lanes. One of 1,2,4,8 and 16.Default is 4.
-b or -baudrate	SRIO Baud rate.One of 1_25G, 2_5G, 3_125G, 5G, 6_25G, 10_3125G. Default is 5G.
-v or -srio_ver	SRIO Specification Version. One of 1_3,2_1,2_2 and 3_0.
-i or -dev_id	SRIO Device ID. One of 8, 16, 32. Default is 8.
-a or -addr_mode	SRIO Addressing Mode. One of 34, 50, 66. Default is 34.
-is or -idle_sel	GEN2 Idle selection.One of 1,2.Default is 2.
-tm or -brc3_traning_mode	GEN3 Training Mode Selection. 0 - Short Run 1 - Long Run.Default - 0.
-rm or -register_model_set	Register model set Selection. 1 - RM-I, 2 - RM-II. Default - 1.
-rs or -seed	Seed value.Default is process id value.
-c or -cov	Enable functional coverage
-help	Shows options.

Log files are stored in logs directory.

9 Tools Used

Tools Used	Version	Vendor	Platform
IUS	Incisive_12.10.020	Cadence	Linux
VCS	G-2012.09-SP1	Synopsys	Linux
Questa	Questa_10.1d	Mentor	Linux
UVM	1.1c	Accellera	Linux