VC Verification IP
PCI Express
SVC User Manual

Version J-2014.12-SP3, July 2015

SYNOPSYS®

Synopsys, Inc.
700 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Preface

About This Manual

This manual contains installation, setup, and usage material for Verilog users of the Synopsys PCIe VIP, and is for design or verification engineers who want to verify PCIe operation using a testbench written in Verilog. Readers are assumed to be familiar with PCIe, object oriented programming (OOP), and Verilog techniques.

Manual Organization

The chapters of this databook are organized as follows:

- Chapter 1, “Introduction”, introduces the Synopsys PCIe VIP and its features.
- Chapter 2, “Installing and Running the PCIe VIP”, describes system requirements and provides instructions on how to install, configure, and begin using the Synopsys PCIe VIP.
- Chapter 3, “Applications”, provides information about the software application layer, where transactions are handled.
- Chapter 4, “Transaction Layer”, describes the layer that encapsulates transactions into TLPs, performs traffic class to virtual channel mapping, controls the flow of transactions, and checks and enforces TLP ordering rules.
- Chapter 5, “Data Link Layer”, describes the layer that processes TLPs from the Transaction Layer or sends TLPs to the Transaction Layer.
- Chapter 6, “Physical Layer”, describes the layer that contains the LTSSM, performs data scrambling and descrambling, lane deskewing and interfacing with the link for transmit/receive data.
- Chapter 7, “Physical Coding Sublayer (2.5Gb / 5Gb/ 8Gb)”, provides information about the Physical Coding Sublayer (PCS). The PCS performs symbol lock at 2.5GT/s, 5GT/s, and 8GT/s., does 8b/10b encoding at 2.5GT/s and 5GT/s and inserts data sync headers at 8GT/s. The PCS also contains the elastic buffer which inserts/deletes SKP symbols from skip ordered sets.
- Chapter 8, “Serializer/Deserializer (pciesvc_serdes.v)”, describes the component that converts the serial bit stream into a stream of 10-bit bytes, and does signal-level validation, receiver clock-recovery and bit alignment.
- Chapter 9, “PCIE Verification Tools”, describes the component that provides a shadow of the DUT’s PCI Express address space for Write and Read transaction checking. The Global PCIE Address Space Shadow is an optional component.
- Chapter 10, “Functional Coverage”, provides information about callback routines that you can use for functional coverage.
- Chapter 11, “PCIe VIP Verification Plans”, provides information about predefined verification plans that you can use in the VCS Verification Planner.

- Chapter 12, "Debugging the PCIe VIP", describes how to use waveform viewers, log files, and the Protocol Analyzer to analyze and resolve problems with your tests.
- Chapter 13, "Memory Model Utility", describes the component that provides a mechanism to manage data, store data structures, and pass arguments in a Verilog environment.
- Chapter 14, "Message Logging Facility", provides information about the facility that enables all components in the VIP to use a common logging facility.
- Chapter A, "Media Module", describes the facility that enables backing up to disk, tape, or other media.
- Appendix B, "Protocol Checks", lists the protocol checks that are performed automatically by the VIP.
- Appendix C, "PCIe UVM Interface", describes the organization and usage of the PCIe UVM interface.
- Appendix E, "PCIe Verilog Testbench Example", provides instructions for downloading, installing, and running a Verilog example of the VIP.
- Appendix G, "Implemented ECNs", contains a list of Engineering Change Notices for the current version of the PCIe VIP.
- Appendix H, "Reporting Problems", provides information about how to get help from Synopsys for problems with the product.

Web Resources

- Documentation through SolvNet: https://solvnet.synopsys.com (Synopsys password required)
- Synopsys Common Licensing (SCL): http://www.synopsys.com/keys

Customer Support

To obtain support for your product, choose one of the following:

- Open a Case through SolvNet.
  - Go to https://onlinecase.synopsys.com/Support/OpenCase.aspx and provide the requested information, including:
    - Product: **Synopsys ...**
    - Sub Product: **...**
    - Tool Version: **...**
    - Fill in the remaining fields according to your environment and your issue.
  - If applicable, provide the information noted in Appendix H, "Reporting Problems".
- Send an e-mail message to support_center@synopsys.com.
  - Include the Product name, Sub Product name, and Tool Version (as noted above) in your e-mail so it can be routed correctly.
  - If applicable, provide the information noted in Appendix H, "Reporting Problems".
- Telephone your local support center.
  - North America:

    Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.

- All other countries:

  http://www.synopsys.com/Support/GlobalSupportCenters

1 Introduction

The Synopsys PCIe Verification IP package is constructed as a collection of Verilog tasks organized in layer specific modules. The modules themselves do not provide the wiring as in a design, but rather serve to create instantiations of a device and hold common states for a given instantiation. The VIP is constructed in a verification language-neutral manner using current state-of-the-art verification techniques such as constrained randomization, transaction and checking modules. Using tasks and passing arguments allows greater flexibility for reuse of the functionality in the test bench and in the test cases. By providing the arguments on the fly to the tasks, the values passed in can be arbitrarily changed by a directed test case. Using tasks also provides for processing in zero time, so that an VIP can be as fast or slow in response as desired.

The functionality within the various layers of the PCIe VIP is compliant to the following specifications:

PCI Express Base 2.1 (supported by the PCIe Gen2 VIP)

PCI Express Base 3.0 (supported by the PCIe Gen3 add-on VIP))

The functionality within the Physical Layer of the PCIe VIP is compliant to the following specifications:

PIPE 2.1 (Support for Gen1 and Gen2 speeds only)

PIPE 4.0 (Support for Gen1, Gen2, and Gen3 speeds)

1.1 Overview of Models

The PCIe VIP includes a Root Complex model and an Endpoint model. The Root Complex and Endpoint models each contain an instantiated Port model as well as appropriate requester and completer applications.

The PCIe Port (MAC) model consists of components that mirror the functionality of the layers defined in the PCIe standard: a Transaction Layer, a Data Link Layer, a Physical Layer (Phy Layer), and 1-to-*n* PCS and SERDES (see Figure 1-1). The Transaction Layer takes commands from the application layer and translate them into Transaction Layer Packets (TLPs). The Transaction Layer is also responsible for monitoring available credits and managing the traffic class and virtual channel queues. The Data Link Layer adds framing and CRC to the TLPs and also generates and receives Data Link Layer Packets (DLLPs). The Physical Layer contains the LTSSM and is responsible for link training. The Physical Layer is also used for assembling and disassembling packets to and from the multiple lanes, performing lane deskew, scrambling and descrambling, and sync header insertion. The PCS model has an Intel PIPE interface, contains an 8b/10b decoder, detects symbol errors, and contains an elastic FIFO for skip insertion and deletion.

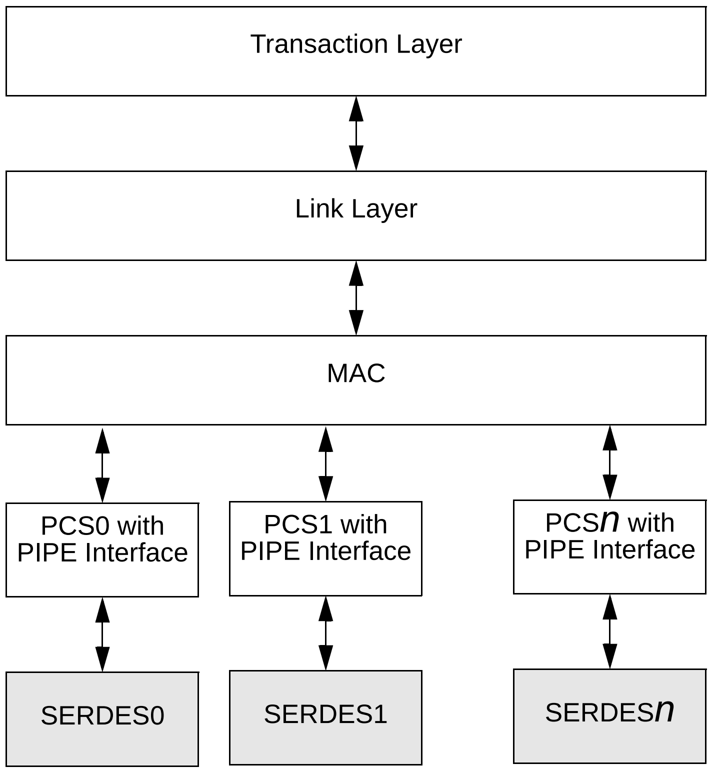


Figure 1-1 PCIe port model with optional SERDES

A Root Complex or Endpoint mode contains application layers as well as a port model (see Figure 1-2).

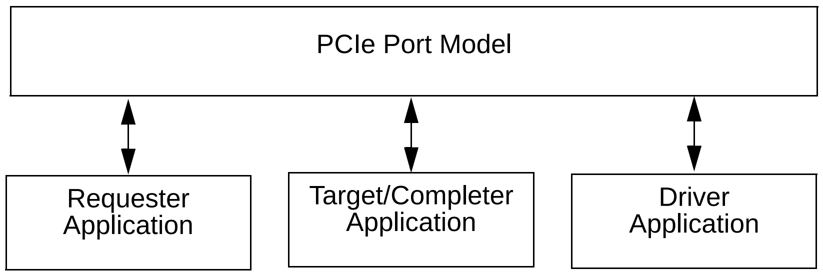


Figure 1-2 Port model with application layer

Multiple Endpoint models can be instantiated along with a root complex model and switch DUT (see Figure 1-3).

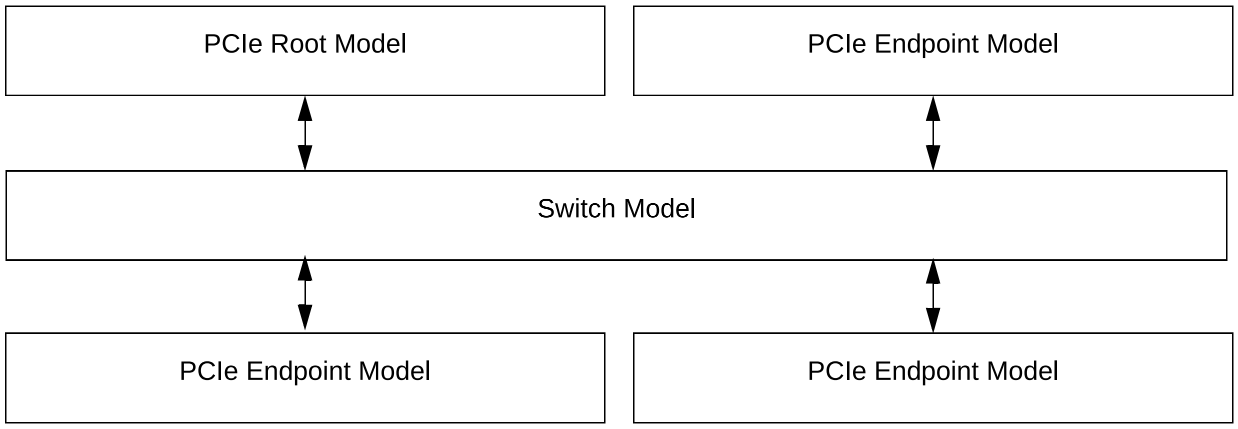


Figure 1-3 A 2x2 PCIe switch with four PCIe port models

1.2 PCIe Device and MAC Model Instantiation

The PCIe device model contains the connectivity, clocking, and basic configuration for the application layer, Transaction Layer, Data Link Layer, and Physical Layer. The MAC model is instantiated internally within the device model.

Generally customers should use the device model.

The device model is generic – it can be personalized to be either a Root Complex or an Endpoint.

1.2.1 Model Names

Models are briefly described in Table 1-1.

Table 1-1 Descriptions of models

Model Name	**Description**
pciesvc_device_serdes_x*n*_model	This is a full device model that connects via a serdes interface and has support for a maximum of *n* lanes (they do not all have to be used).
pciesvc_device_pma_x*n*_model	This is a full device model that connects via 10bit PMA interface and has support for a maximum of *n* lanes (they do not all have to be used).
pciesvc_device_mpipe_x*n*_model	This is a full device (rc or endpoint) model that has a standard master PIPE interface to a DUT Phy.
pciesvc_device_spipe_x*n*_model	This is a full device (rc or endpoint) model that has a slave PIPE interface designed to connect directly to a DUT MAC master PIPE interface.
pciesvc_mac_mpipe_x*n*_model	This is a MAC only model (no applications) for use in the compliance environment. It is also automatically instantiated within the full device model.
pciesvc_mac_spipe_x*n*_model	This is a MAC only model (no applications) for use in the compliance environment. It is also automatically instantiated within the full device model.
pciesvc_phy_serdes_x*n*_model	This is a phy model used only in the compliance environment.

NOTE: Models supporting 8G have "_8g" appended to the model name.

1.2.2 Model Wire Interface Options

The VIP is connected to a DUT via either a Serdes, 10-bit PMA, or PIPE (both “master” and “slave”) interface. The PIPE interface supports PIPE specification versions 2 or 3 depending on the model used.

Table 1-2 defines the port list for each interface configuration of the pciesvc_model instantiation.

Reset to all VIP models is active high. Reset must be de-asserted at time 0 such that a posedge is seen by the VIP. Reset must be asserted for exactly 100ns. Once de-asserted, it should remain de-asserted for the remainder of the simulation. DO NOT attempt to assert VIP reset to re-initialize the VIP. Do not attempt to configure the VIP until the reset has been de-asserted.

Note: All unused input ports must be tied to 0, with the exception of *_elec_idle_*n*. Any unused *_elec_idle_*n* input ports must be tied to a 1.

Table 1-2 pciesvc_model ports

Model Interface Type		**I/O**	**Description**
Serdes Interface.	Reset		Active high reset. Must only be asserted for 100ns ONCE per simulation at the beginning.
	phy*n*_rx_datap	I	Serial datap in to VIP
	phy*n*_rx_datan	I	Serial datan in to VIP
	phy*n*_tx_datap	I	serial datap out of VIP
	phy*n*_tx_datan	I	serial datan out of VIP

Table 1-2 pciesvc_model ports (Continued)

Model Interface Type		I/O	Description
“Master” PIPE Version 3 Interface This type of interface is a standard PIPE interface to a phy. PIPE 3 models do not have a _8g postfix in their file names.	reset	I	Active high reset. Must only be asserted for 100ns ONCE per simulation at the beginning.
	pipe_reset_*n*	O	pipe reset signal to phy
	pipe_clk	I	pipe clock from phy
	powerdown[1:0]	O	Command to change power state of the phy
	rate[1:0]	O	Link signaling rate control
	txdetectrx	O	Command to detect receiver
	phy_status[31:0]	I	Phy response to rate or power change. For pipe 2, only bit 0 is used.
	data_bus_width[1:0]	I	Indicates the per lane pipe data bus width
	rx_data_*n*[31:0]	I	Per lane receive data into the VIP. Supports 8, 16, and 32bit bus widths
	rx_data_k_*n*[3:0]	I	Per lane control indication. Supports 8, 16, and 32 bit bus widths.
	rx_status_*n*[2:0]	I	Per lane receiver status. See PIPE spec for encodings. Note: Any unused ports must be tied to a 0.
	rx_valid_*n*	I	Per lane indication that data coming into the VIP is valid and symbol lock has been achieved.
	rx_data_valid_*n*	I	Per lane indication that data coming into the VIP is valid. Used primarily for rate matching. NOTE: pipe 3 only.

Table 1-2 pciesvc_model ports (Continued)

Model Interface Type		I/O	Description
	rx_elec_idle_*n*	I	Per lane indication that electrical idle is being received into the VIP Note: Any unused ports must be tied to a 1.
	invert_rx_polarity_*n*	O	Per lane indication to perform polarity inversion on data into the VIP
	tx_data_*n*[31:0]	O	Per lane transmit data out from the VIP
	tx _data_k_*n*[3:0]	O	Per lane transmit_control from the VIP
	tx _ei_code_*n*[31:0]	O	Per lane error injection code. Indicates to the phy that an EI was either injected, or phy needs to inject the EI indicated by the code. For VIP only. No connection to DUT.
	tx_compliance_*n*	O	Per lane transmit compliance pattern enable from the VIP
	tx_elect_idle_*n*	O	Per lane transmit electrical idle/transmit enable from the VIP

Table 1-2 pciesvc_model ports (Continued)

Model Interface Type		I/O	Description
"Slave" PIPE Version 3 Interface	reset	I	Active high master reset. Must only be asserted for 100ns ONCE per simulation at the beginning.
The signals prefixed "attached_" are relative to the DUT and are designed to connect up in a slave-like manner to a MAC DUT. PIPE 3 models do not have a _8g postfix in their file name.	attached_pipe_reset_*n*	I	pipe reset signal from the mac.
	pipe_clk	O	pipe clock generated by the pipe slave for use by the DUT mac.
	attached_powerdown[1:0]	I	Command from the DUT MAC to change power state of the phy.
	attached_rate[1:0]	I	Link signaling rate control from the DUT MAC. NOTE: Port width is 1 bit for pipe2 and 2 bits for pipe3.
	attached_txdetectrx	I	Command from the DUT mac to detect receiver.
	attached_block_align_control	I	Controls slave block alignment. NOTE: pipe 3 only.
	attached_phy_status[31:0]	O	Phy response to rate or power change. Bit 0 only used for pipe2.
	attached_data_bus_width[1:0]	O	Defines the bus width of per lane data.
	attached_rx_data_*n*	O	Per lane receive data into the DUT. Supports 8, 16, and 32bit bus widths
	attached_rx_data_k_*n*	O	Per lane control indication. Supports 8, 16, and 32 bit bus widths.
	attached_rx_status_*n*[2:0]	O	Per lane receiver status. See PIPE spec for encodings.
	attached_rx_valid_*n*	O	Per lane Iindication that data coming into the DUT is valid and symbol lock has been achieved.
	attached_rx_elec_idle_*n*	I/O	Per lane indication that electrical idle is being received into the DUT

Table 1-2 pciesvc_model ports (Continued)

Model Interface Type		I/O	Description
"Slave" PIPE Version 3 Interface (continued)	attached_invert_rx_polarity_*n*	I	Per lane indication to perform polarity inversion on data into the DUT
	attached_tx_data_*n*[31:0]	I	Per lane transmit data out from the DUT.
	attached_tx _data_k_*n*[3:0]	I	Per lane transmit_control from the DUT.
	attached_tx_ei_code_*n*[31:0]	I	Per lane error injection code. Indicates to the phy that an EI was either injected, or phy needs to inject the EI indicated by the code. For VIP only. No connection to DUT.
	attached_tx_compliance_*n*	I	Per lane transmit compliance pattern enable from the DUT
	attached_tx_elect_idle_*n*	I	Per lane transmit electrical idle/transmit enable from the DUT Note: Any unused ports must be tied to a 1.
"Master" PIPE Version 4 Interface This type of interface is a standard PIPE interface to a phy. The 8G models require a PIPE4 interface, hence all models in the InstantionModels directory with the _8g suffix use the PIPE4.	reset	I	Active high reset. Must only be asserted for 100ns ONCE per simulation at the beginning.
	pipe_reset_*n*	O	Pipe reset signal to phy
	pipe_clk	I	Pipe clock from phy
	powerdown[2:0]	O	Command to change power state of the phy No vendor specific power levels are supported.
	rate[1:0]	O	Link signaling rate control
	pclk_rate[2:0]	O	Specifies the value of the pipe clock.
	txdetectrx	O	Command to detect receiver
	block_align_contro	O	Controls slave block alignment. NOTE: Pipe 3 only.
	tx_margin[2:0]	O	Selects transmitter voltage levels
	tx_swing	O	Controls transmitter voltage swing level.
	lf[5:0]	O	Provides the LF value advertised by the link partner.
	fs[5:0]	O	Provides the full swing value advertised by the link partner.
	width[1:0]	O	Outputs the width of the per-lane data bus the mac is currently using.
	rx_standby[31:0]	O	Controls whether the phy RX is active when in L0 or L0s.
	phy_status[31:0]	I	Phy response to rate or power change. For pipe 2, only bit 0 is used.

Table 1-2 pciesvc_model ports (Continued)

Model Interface Type		I/O	Description
	rx_standby_status[31:0]	I	Reflects the active/standby state of the rx receiver.
	data_bus_width[1:0]	I	Indicates the per lane pipe data bus width.
	rx_data_*n*[31:0]	I	Per lane receive data into the SVC. Supports 8, 16, and 32bit bus widths.
	rx_data_k_*n*[3:0]	I	Per lane control indication. Supports 8, 16, and 32 bit bus widths.
	rx_status_*n*[2:0]	I	Per lane receiver status. See PIPE spec for encodings. Note: Any unused ports must be tied to a 0.
	rx_valid_*n*	I	Per lane indication that data coming into the SVC is valid and symbol lock has been achieved.
	rx_data_valid_*n*	I	Per lane indication that data coming into the SVC is valid. Used primarily for rate matching.
	rx_elec_idle_*n*	I	Per lane indication that electrical idle is being received into the SVC. Note: Any unused ports must be tied to a 1.
	rx_start_block_*n*	I	Per lane indication that electrical idle is being received into the SVC is the first byte of data in the block.
	rx_sync_header_*n*	I	Per lane block sync header.
	invert_rx_polarity_*n*	O	Per lane indication to perform polarity inversion on data into the SVC.
	tx_data_*n*[31:0]	O	Per lane transmit data out from the SVC.
	tx _data_k_*n*[3:0]	O	Per lane transmit_control from the SVC
	tx _ei_code_*n*[31:0]	O	Per lane error injection code. Indicates to the phy that an EI was either injected, or phy needs to inject the EI indicated by the code. For SVC only. No connection to DUT.
	tx_compliance_*n*	O	Per lane transmit compliance pattern enable from the SVC.
	tx_elect_idle_*n*	O	Per lane transmit electrical idle/transmit enable from the SVC.
	tx_data_valid_*n*	O	Per lane indication that data transmitted by the SVC is valid. Used primarily for rate matching.

Table 1-2 pciesvc_model ports (Continued)

Model Interface Type		I/O	Description
	tx_start_block_*n*	O	Per lane indication that data transmitted by the SVC is the first byte of data in the block.
	tx_sync_header_*n*[1:0]	O	Per lane block sync header.
	local_tx_preset_coefficients*n*[17:0]	I	Coefficients returned from a coefficient lookup request.
	link_eval_feedback_figure_of_merit*n*[7:0]	I	Figure of merit value from a link equalization evaluation request.
	link_eval_feedback_direction_change*n*[5:0]	I	Coefficient direction feedback from rx link eval request.
	local_fs*n*[5:0]	I	Provides the FS value for the phy.
	local_lf*n*[5:0]	I	Provides the LF value for the phy.
	local_tx_coefficients_valid*n*	I	Indicates that the values in local_tx_preset_coefficients[] are valid.
	tx_deemph*n*[17:0]	O	Selects transmitter deemphasis.
	local_preset_index*n*[3:0]	O	Index for the local phy preset coefficients requested by the mac.
	rx_preset_hint*n*[2:0]	O	RX preset hint for the receiver.
	get_local_preset_coefficients*n*	O	Request a preset to coefficient mapping lookup.
	rx_eq_eval*n*	O	Request from the mac for the receiver to perform an equalization evaluation.
	invalid_request*n*	O	Indicates the link eval feedback requested was out of range

Table 1-2 pciesvc_model ports (Continued)

Model Interface Type		I/O	Description
"Slave" PIPE Version 4 interface The signals prefixed "attached_" are relative to the DUT and are designed to connect up in a slave like manner to a MAC DUT. The 8G models require a PIPE4 interface, hence all models in the InstantionModels directory with the _8g suffix use the PIPE4.	reset	I	Active high master reset. Must only be asserted for 100ns ONCE per simulation at the beginning.
	attached_pipe_reset_*n*	I	pipe reset signal from the mac.
	pipe_clk	O	pipe clock generated by the pipe slave for use by the DUT mac.
	max_pclk	O	Maximum pclk rate for a given speed.
	attached_powerdown[2:0]	I	Command from the DUT MAC to change power state of the phy.
	attached_rate[1:0]	I	Link signaling rate control from the DUT MAC. NOTE: port width is 1 bit for pipe2 and 2 bits for pipe3.
	attached_pclk_rate[2:0]	I	PCLK rate requested by the mac.
	attached_txdetectrx	I	Command from the DUT mac to detect receiver.
	attached_block_align_control	I	Controls slave block alignment. NOTE: pipe 3 only.
	attached_tx_margin[2:0]	I	Selects transmitter voltage levels.
	attached_tx_swing	I	Controls transmitter voltage swing level.
	attached_lf[5:0]	I	Provides the LF value advertised by the link partner.
	attached_fs[5:0]	I	Provides the full swing value advertised by the link partner.
	attached_width[1:0]	I	Reflects the width of the data bus the mac wants to use.
	attached_rx_standby[31:0]	I	Controls whether the phy RX is active when in L0 or L0s.
	attached_phy_status[31:0]	O	Phy response to rate or power change. Bit 0 only used for pipe2.
	attached_rx_standby_status[31:0]	O	Reflects the active/standby state of the rx receiver.
	attached_data_bus_width[1:0]	O	Defines the bus width of per lane data.
	attached_rx_data_*n*[	O	Per lane receive data into the DUT. Supports 8, 16, and 32bit bus widths.
	attached_rx_data_k_*n*[	O	Per lane control indication. Supports 8, 16, and 32 bit bus widths.
	attached_rx_status_*n*[[2:0]	O	Per lane receiver status. See PIPE spec for encodings.

Table 1-2 pciesvc_model ports (Continued)

Model Interface Type		I/O	Description
	attached_rx_valid_*n*[	O	Per lane Iindication that data coming into the DUT is valid and symbol lock has been achieved.
	attached_rx_data_valid_*n*[	O	Per lane Iindication that data coming into the DUT is valid. Used primarily for rate matching. NOTE: pipe 3 only.
	attached_rx_elec_idle_*n*	IO	Per lane indication that electrical idle is being received into the DUT
	attached_rx_start_block_*n*[	O	Per lane Iindication that data transmitted by the DUT is the first byte of data in the block. NOTE: pipe 3 only.
	attached_rx_sync_header_*n*[[1:0]	O	Per lane block sync header. NOTE: pipe 3 only.
	attached_invert_rx_polarity_*n*[	I	Per lane indication to perform polarity inversion on data into the DUT.
	attached_tx_data_*n*[[31:0]	I	Per lane transmit data out from the DUT.
	attached_tx _data_k_*n*[[3:0]	I	Per lane transmit_control from the DUT.
	attached_tx_ei_code_*n*[[31:0]	I	Per lane error injection code. Indicates to the phy that an EI was either injected, or phy needs to inject the EI indicated by the code. For SVC only. No connection to DUT.
	attached_tx_compliance_*n*[	I	Per lane transmit compliance pattern enable from the DUT
	attached_tx_elect_idle_*n*[	I	Per lane transmit electrical idle/transmit enable from the DUT.Note: Any unused ports must be tied to a 1.
	attached_tx_data_valid_*n*[	I	Per lane Iindication that data transmitted to the SVC is valid. Used primarily for rate matching. NOTE: pipe 3 only.
	attached_tx_start_block_*n*[	O	Per lane Iindication that data transmitted to the SVC is the first byte of data in the block. NOTE: pipe 3 only.
	attached_tx_sync_header_*n*[[1:0]		Per lane block sync header. NOTE: pipe 3 only.
	attached_local_tx_preset_coefficients*n* [[17:0]		Coefficients returned from a coefficient lookup request.
	attached_link_eval_feedback_figure_of_merit*n*[7:0]	O	Figure of merit value from a link equalization evaluation request..
	attached_link_eval_feedback_direction_change*n*[5:0]	O	Coefficient direction feedback from rx link eval request.

Table 1-2 pciesvc_model ports (Continued)

Model Interface Type		I/O	Description
	attached_local_fs*n*[5:0]	O	Provides the FS value for the phy.
	attached_local_lf*n*[5:0]	O	Provides the LF value for the phy.
	attached_local_tx_coefficients_valid*n*	O	Indicates that the values in local_tx_preset_coefficients[] are valid.
	attached_tx_deemph*n*[17:0]	I	Selects transmitter deemphasis.
	attached_local_preset_index*n*[3:0]	I	Index for the local phy preset coefficients requested by the mac.
	attached_rx_preset_hint*n*[2:0]	I	RX preset hint for the receiver.
	attached_get_local_preset_coefficients *n*	I	Request a preset to coefficient mapping lookup.
	attached_rx_eq_eval*n*	I	Request from the mac for the receiver to perform an equalization evaluation.
	attached_invalid_request*n*	I	Indicates the link eval feedback requested was out of range.

1.2.3 Configuring the PIPE Data Bus Width

For PIPE4 users, the per-lane data bus width and rate of the pipe clock are determined by the pipe signals pclk_rate and width (not to be confused with the pipe signal data_bus_width). If you are using the SPIPE model, simply connect up the pclk_rate and width pipe signals to your MAC. If you are using the MPIPE model, use the ConfigurePipeWidthPclkTable task call, in order for the mac to properly drive pclk_rate and width. See Table 6-2 for details about the ConfigurePIpeWidthPclkTable task.

The pipe models that do not end with "_8g" support PIPE 2.1/PIPE 3.0. In these versions the pclk_rate signal does not exist, and the ConfigurePipeWidthTable task should be called on the spipe model to tell it at what width to operate for various speeds. See "Physical Layer" for more details. The mpipe model for PIPE2.1/PIPE3.0 takes the width reflected on the data_bus_width signal. Refer to PIPE2.1/PIPE3.0 for more information on the data_bus_width signal.

If you are using the SERDES model or the PMA model, do not use the ConfigurePipeWidthTable task or the ConfigurePipeWidthPclkTable task.

1.2.4 Configuring the PIPE Data Bus Width

For PIPE4 users, the per-lane data bus width and rate of the pipe clock are determined by the pclk_rate and width pipe signals, (not to be confused with the pipe signal data_bus_width). If you are using the SPIPE model you simply need to connect up the pclk_rate and width pipe signals to their MAC. MPIPE users need to call the ConfigurePIpeWidthPclkTable task in order for the mac to properly the drive pclk_rate and width signals. See "Physical Layer Configuration Tasks" for information about the ConfigurePipeWidthPclkTable task.

The pipe models that do not end with _8g support PIPE 2.1/PIPE 3.0. In these versions of the pipe spec the mac does not specify the width and rate of the pipe clock, so you should call the ConfigurePipeWidthTable task on the spipe model to tell it the width at which to operate at various speeds. See "Physical Layer

Configuration Tasks" for more details. The mpipe model for PIPE2.1/PIPE3.0 will take the width reflected on the data_bus_width signal. See Table 1-2 for information about the data_bus_width signal.

If you are using the SERDES or PMA models, do not use the ConfigurePipeWidthTable task or the ConfigurePipeWidthPclkTable task.

1.2.5 Model Configuration Overview

It is recommended that configuration tasks be used after reset is de-asserted from the pciesvc_model.

Recommended tasks to be set prior to usage are described in Table 1-3. The details of the task may be found in the respective section of the documentation.

Table 1-3 Configuration tasks to set before usage

Name	Layer	Description
SetAllocatedCredits	Transaction Layer	Set Initial credits for a VC
SetVcEnable	Transaction Layer	Enable / Disable VCs
SetTrafficClassMap	Transaction Layer	Maps TCs to VCs
AddMemAddrApplIdMapEntry	Transaction Layer	If not using the default, sets the Application ID for the memory target address range
AddIOAddrApplIdMapEntry	Transaction Layer	If not using the default, sets the Application ID for the I/O target address range
SetLinkEnable	Data Link Layer	Enable / disable the Data Link Layer
SetSupportedSpeeds	Physical Layer	Sets the supported speeds that will be advertised during link training. If link training is bypassed, the highest supported speed will be used.
SetLinkWidth	Physical Layer	Sets the width of the link, enable the state machine to start

1.2.6 Model Parameters

Several parameters are set at the model. They typically 'trickle-down' to the individual layers. Table 1-4 lists those parameters.

Table 1-4 Parameters set in the model

Parameter Name	Settable VAR?*	Type	Range	Default Value	Description
DEVICE_IS_ROOT	No	Integer	0-1	(Per model)	This value is '1' if the particular model is for a root, else it is '0'.
NUM_PMA_I NTERFACE_BITS	No	Integer	10, 16, 32, 64, 128	10	Number of bits on the PMA interface

Table 1-4 Parameters set in the model (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default Value	Description
PCIE_SPEC_VER	No	Real	1.1, 2.0, 2.1, 3.0	PCIE_SPEC_VER_3_0	See Include/pciesvc_parms.v: PCIE_SPEC_VER_* Note: Please set this here, not in the individual layers.
HIERARCHY_NUMBER	No	Integer	0 - large value	0	The per-root hierarchy number – these start at 0 and count upwards. Set this to the root hierarchy that this model belongs to.
ENABLE_SHADOW_MEMORY_CHECKING	No	Integer	0-1	1	If set, applications will check memory reads against the shadow memory. Make sure this is not enabled if the shadow memory is not instantiated.
DISPLAY_NAME	No	String		"pciesvc_xx_yy_model_zz." (Specific to each model).	String prefixed to messages to display in the output log.
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with "_VAR". This is useful if using higher level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.					

1.2.7 Multiple Root Hierarchies

If you need to instantiate multiple root hierarchies, then each one must be marked uniquely to distinguish it. This is done via the HIERARCHY_NUMBER parameter. For each root (and associated endpoint) model you instantiate, you need to defparam the appropriate hierarchy number to that model. If you are using the Shadow Memory mechanism for ding automatic checking of data, you also need to defparam those shadows respectively (See "The Global PCIE Address Space Shadow").

1.2.8 Fast Link Training

A common way speed up link training is to decrease the number of training sets transmitted in Polling.Active. The VIP supports this option, which is controlled by the Physical Layer parameter NUM_TX_TS1_IN_POLLING_ACTIVE (see Table 6-8 on page 139). This parameter and all LTSSM timeout parameters should be set to match whatever values are used in the DUT in order to obtain valid results during abbreviated link training.

1.3 Test Environment Requirements

This section details requirements for the PCIE model to function properly in the customer's test environment. Refer to the PCIE/Verilog/Example directory for example of model usage.

The global memory svc_mem must be instantiated in the test environment. It is recommended that only one instantiation of the memory be used. For further information regarding the svc_mem module, refer to "Memory Model Utility". To properly use the svc_mem model, the define PCIESVC_MEM_PATH must be set to the full instantiation name of the global memory model.

A reset signal applied to the VIP must be in the reset state at time zero. There is no limit in the length of time the reset is held asserted. However, once the reset is de-asserted, it may not be asserted again. Thus, for proper DUT verification, it is advisable to separate the VIP model reset from the DUT reset

The various parameters described in the following chapters that control the behaviors of the VIP may be controlled using the defparam syntax OR by modifying the associated *_VAR during simulation runtime. The *_VAR complement is commonly used in environments where the VIP is wrapped around an abstract verification language such as SystemC, ***e***, or OpenVera. Verilog only or System Verilog test environments often make use of the *_VAR complement as well. When using the *_VAR feature, a *_VAR may not be assigned until at least 1 simulation time tick has passed as the _VARs will be assigned in the model at time 0 to the value of the corresponding parameter. In the top level example, INVERT_TX_POLARITY in the pcs is assigned after a #1 delay. (Note: An acceptable alternative solution is to add defparam port0.pcs0.INVERT_TX_POLARITY = 1 after a pcie_port_model instantiation).

At the completion of the test, it is useful to output the various statistics kept by the VIP to the log file. This helps in determining corner case coverage and debug info.

A set of SystemVerilog APIs can optionally be used if desired. To include these APIs into the model, you must set the two defines PCIESVC_INCLUDE_SYSTEMVERILOG_API and SVC_INCLUDE_SYSTEMVERILOG_API in order for the SystemVerilog API routines to be pulled into the VIP.

1.3.1 Gen3 Requirements

Enabling Gen3 support in the VIP requires the instantiation of *_8g.v model and EXPERTIO_PCIESVC_INCLUDE_8G defined on the command line. Even if EXPERTIO_PCIESVC_INCLUDE_8G is not defined, *_8g.v models will still work correctly as long as Gen3 functionality is not exercised. Gen3 funtionality includes 8G speeds, L1 substates, and so on.

1.4 System Verilog APIs

The product is methodology neutral, and may be control from any high level language including SystemVerilog.

There are two common SystemVerilog methodology approaches: UVM/OVM/VMM and the “Two Kingdoms” methodology. The PCIe VIP application programming interfaces (APIs) are neutral to both of these approaches.

For test environments that use SystemVerilog constructs such as program blocks and other elements that do not allow direct scope resolution, a set of SystemVerilog class object APIs are provided. The top level class is instantiated, then `new` is called with the hierarchical path to the model that the class should be bound to. The class constructor locates all the layers in the VIP model referred to, and instantiate subclasses per layer in the model.

Once this is done, you may set any *constraint* _VAR in the model by calling a SystemVerilog API method to set or get the constraint value. The method name is formed by:

1. Prepending “Set” or “Get” to the _VAR name.
2. Making the name of the _VAR into mixed “camel” case
3. Removing the “_”.
4. Dropping the _VAR suffix.
5. Appending “Api” to the end.

For example, the Data Link Layer MIN_INITFC_DELAY constraint VAR becomes root0.port[0].dl[0].SetMinInitfcDelayApi(*value*). Although this looks like a hierarchical path, it is actually a navigation through a structure of classes, hence use of the "[]".

Likewise there are SystemVerilog APIs for all user and configuration tasks and functions documented: Just append "Api" to the end of the Verilog task or function listed. For example, the Transaction Layer `SetTrafficClassMap()` task is called via the SystemVerilog root0.port[0].tl[0]SetTrafficClassMapApi().

There is an example of instantiating and using the SV APIs in PCIE/Verilog/Example/test_basic_sv.sv.

1.5 Running Multiple Simulations in the Same Directory

Many environments run multiple simulations; in general this is supported without any issues. However, if you need to run more than one simultaneously in the same directory, the VIP (and, of course, your particular simulator setup) must be configured specifically for this. To enable this configuration in the VIP, set the following plusarg in your simulation:

```
+expertio_unique_sim_str="XXX"
```

Set the string *"XXX"* to a per-simulation-unique string (one possibility is to use the process ID, or a global incrementing number). Note that you need to also take care of other per-simulation information (e.g. make sure log files, etc. are also unique) in your environment.

1.6 Randomization Seeds and Modes

In the SVC, there are two different modes of randomization in terms of how the random seed is used. The seed can either be GLOBAL (global seed mode) or a seed per THREAD (random by thread mode).

In general, global seed mode should be used with Cadence NC and Mentor Questa simulators as it is the easiest method to use. Some simulators (such as Synopsys VCS) do not have predictable thread scheduling, hence requiring a per-thread randomization context to provide repeatability of simulation results.

To utilize a GLOBAL seed in the SVC, define a globally available container for the seed at the top level of your test bench. Then set the +define+SVC_RANDOM_SEED_SCOPE top.global_random_seed. This define must be the fully scoped path to the container holding the random seed. Each `$random` call in the SVC will update the value in the seed container. For example, in top.v:

```
    integer global_random_seed;
```

A second method is also available for utilizing a global seed: instead of the above you can compile and link in a replacement `$random` PLI provided by Synopsys. Synopsys' `$random` holds its own global seed container (that is, it acts like a C random call) unlike the `$random` built into the simulators. It is easy to link in this replacement. Contact Synopsys at support_center@synopsys.com for the instructions if interested in this.

To use random by THREAD mode, set the define SVC_RANDOM_BY_THREAD. This saves a unique copy of the seed for every included instance of the svc_util_tasks.v file by a module, hence there will be a unique seed per Verilog `always` thread/module The SVC_RANDOM_SEED define is also usable in a per-thread randomization context (and will be used for all contexts). If you are using RANDOM_BY_THREAD, you will probably want to set the SVC_RANDOM_SEED define to a PLI call to discover the scope of the caller and handle setting unique seeds per thread. The default for this define is to not be set, in which case the randomization method will be the normal GLOBAL random seed method.

1.7 SVC Memory Usage

TLPs are passed with pointers to or from the SVC via the Verilog and SystemVerilog interfaces. The pointers reference a memory location in the svc_mem. The svc_mem instantiation is discussed in "Test Environment Requirements", and the usage is described in "Memory Model Utility".

To store a TLP in the svc_mem, the memory must first be allocated via AllocMemory. The pointer (address) that is returned is used to store the TLP via WriteMemory task. The TLP can be passed to the SVC using that pointer. For TLPs received from the SVC, the TLP can be accessed in svc_mem via a ReadMemory task call. The pointer to the TLP will be provided as a task argument (*ptr) in the SVC interface task call. Once the TLP is consumed, the memory space must be returned to the free pool via a FreeMemory task call to the svc_mem. This applies to both sent and received TLPs.

The UVM interface does not use the svc_mem to pass TLPs: it uses dynamic arrays defined in the sequence item.

NOTE: Neglecting free memory will result in the svc_mem being completely consumed. If this occurs, any one of the AllocMemory task calls within the SVC may result in an error. The simulation will stop at this point. If this occurs, make sure svc_mem is freed for all TLPs sent to or received from the SVC. See "Driver Memory Management Requirements" for further details about driver requirements.

1.8 Compliance Patterns

The compliance and modified compliance patterns defined in the PCIE specification contain sequences of data that would be considered an error during normal operation. For example, at 2.5G and 5G part of the compliance pattern is to send a COM followed by data symbols that do not make a legal ordered set.

At 8G there are ordered set blocks filled with symbols that do not make up a valid ordered set. Additionally SKP ordered sets at 8G contain data associated with compliance rather than the contents of the LFSR. Because there is no way to know exactly when the DUT starts transmitting the compliance pattern vs normal link training, the VIP can and will likely flag some ordered set violations until it recognizes the compliance pattern.

This means that when the vip initially receives the compliance pattern or modified compliance pattern, the user will be required to suppress or demote some error messages until the VIP obtains lock on the pattern in order to obtain a passing test. In PIPE simulations, the VIP should recognize a compliance or modified compliance pattern by the time the pattern has completed its first cycle.

In serial simulations it will longer for the VIP to recognize compliance, because if a speed change occurs in polling.compliance, then the VIP must acquire bit lock and symbol/block alignment first. The modified compliance pattern at 8G in serial mode will take an especially long time, because the EIEOS required for block alignment occurs only once every 65792 blocks.

2
Installing and Running the PCIe VIP

This chapter provides information on how to install and set up the VC VIP for PCI Express.

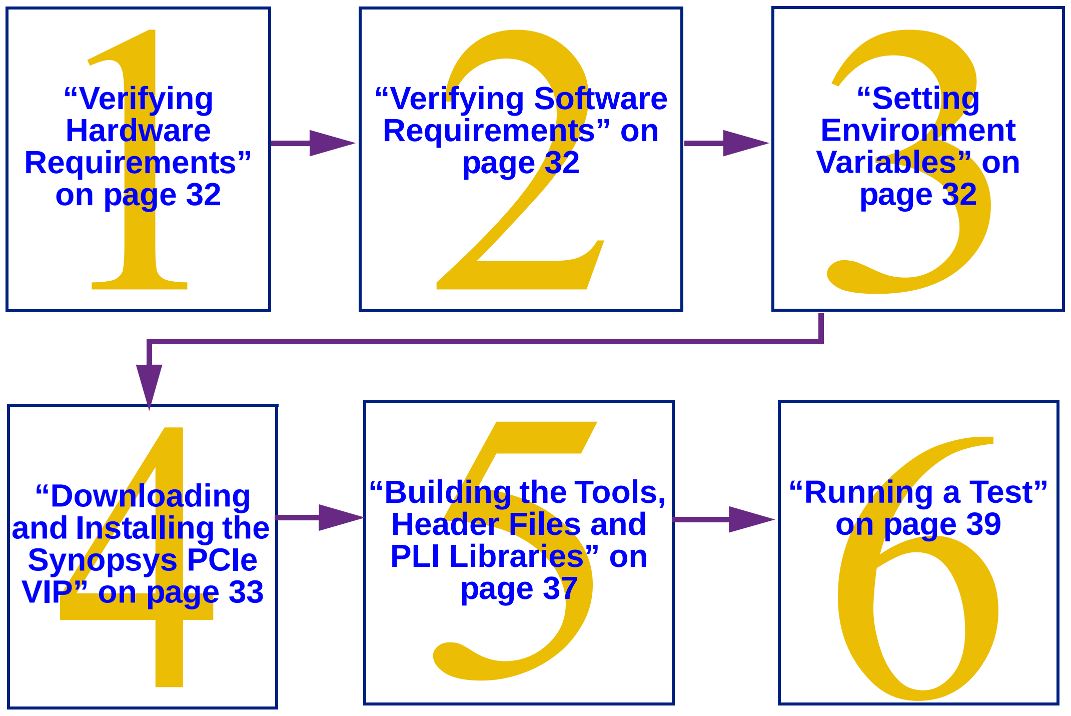
2.1 Accessing Product Documentation

After the PCIe VIP is downloaded and installed, the documentation resides at:

$EXPERTIO_PCIESVC_INSTALL_PATH/doc.

2.2 Process Overview

To install PCIe VIP, you must follow the steps in this installation guide.


1
"Verifying Hardware Requirements" on page 32
2
"Verifying Software Requirements" on page 32
3
"Setting Environment Variables" on page 32
4
"Downloading and Installing the Synopsys PCIe VIP" on page 33
5
"Building the Tools, Header Files and PLI Libraries" on page 37
6
"Running a Test" on page 39

2.3 Verifying Hardware Requirements

Table 2-1 describes the system requirements for the VC VIP for PCI Express product.

Table 2-1 System Requirements

Element	Minimum System Requirement
Platform/OS and VCS	See "Platform/OS and Simulator Software" on page 32
Disk Space	• Approximately 20 MB available disk space for installation and basic use
Memory	• For users of VCS or VCS MX, see VCS Installation Notes or VCS MX Installation Notes, which are accessible from the Synopsys Installation Guide (search for "synopsys installation guide" on www.synopsys.com). • For users of other simulators, see the documentation for your simulator.

Note You must have access to SolvNet to download the release image. You can sign up to receive updates for any VIP component through this web site as well.

2.4 Verifying Software Requirements

The PCIe VIP is tested with certain versions of platforms and simulators. This section lists software that the PCIe VIP requires.

2.4.1 Platform/OS and Simulator Software

To see which platform/OS and simulator versions are qualified for use with the PCIe VIP, check the support matrix in the *Verification IP for PCIe VIP Release Notes*.

2.4.2 Synopsys Common Licensing (SCL) Software

The SCL software provides the licensing function for the PCIe VIP. Acquiring the SCL software is covered in "Licensing" on page 39.

2.4.3 Other Third Party Software

- **Adobe Acrobat**: PCIe VIP documents are available in Acrobat PDF files. You can get Adobe Acrobat Reader for free from http://www.adobe.com.
- **HTML browser:** The VC VIP for PCI Express includes class reference documentation in HTML. The following browser/platform combinations are supported:
  - Microsoft Internet Explorer 6.0 or later (Windows)
  - Firefox 1.0 or later (Windows and Linux)
  - Netscape 7.x (Windows and Linux)

2.5 Setting Environment Variables

2.5.1 Setting Common Environment Variables

Set the ARCH and SVC_SIM variables as shown below:

EXPERTIO_PCIESVC_INSTALL_PATH

Set this variable to *installationPath*/PCIE/Verilog.

ARCH

Set this variable to your architecture. (In a Cadence environment, use the the UNIX `uname` command. If `uname` returns Linux or one of its variants, set ARCH to lnx86. If `uname` returns Solaris set it to sun4v.)

Supported values:

lnx86 (Linux on x86)

sun4v (sun-solaris)

Default: none

SVC_SIM

Set to one of the following:

VCS (Synopsys simulators: VCS, VCS-MX)

NC (Cadence simulators: NC-Verilog, IUS)

MT (Mentor simulators: MTI, Questa)

2.5.1.1 Simulator-Specific Variables and Command Options

The installation helper scripts (`vcs`, `ncs`, or `mts` – see “Helper Scripts”) attempt to set the command line appropriately, but you need to set environment variables as described below.

2.5.1.1.1 VCS/VCS MX Environment Variable

VCS_HOME	Set to point to your local Synopsys installation tree Default: none

2.5.1.1.2 MTI/Questa Environment Variable

MT_INST_DIR	Set to point to your local MTI/Questa installation tree Default: none

2.5.1.1.3 NC Environment Variables

CDS_INST_DIR	Set to point to your local Cadence installation tree. Default: none
LD_LIBRARY_PATH	Path to your shared libraries. Both the Cadence shared libraries and those that live under the Verilog sub-directory in the VIP installation directory are needed. Default: none

2.6 Downloading and Installing the Synopsys PCIe VIP

Attention The Electronic Software Transfer (EST) system only displays products your site is entitled to download. If the product you are looking for is not visible please contact est-ext@synopsys.com.

Follow the instructions below for downloading the software from Synopsys. You can download from the Download Center using either HTTPS or FTP, or with a command-line FTP session. If your Synopsys SolvNet password is unknown or forgotten, go to http://solvnet.synopsys.com.

Passive mode FTP is required. The passive command toggles between passive and active mode. If your FTP utility does not support passive mode, use http. For additional information, refer to the following web page:

https://www.synopsys.com/apps/protected/support/EST-FTP_Accelerator_Help_Page.html

2.6.1 Downloading From the Electronic Software Transfer (EST) System (Download Center)

1. Point your web browser to "https://solvnet.synopsys.com/DownloadCenter".
2. Enter your Synopsys SolvNet username.
3. Enter your Synopsys SolvNet password.
4. Click the "Sign In" button.
5. Choose the "ExpertIO VIP" from the list of available products under "My Product Releases".
6. Select the product version from the list of available versions.
7. Click the "Download Here" button for HTTPS download.
8. After reading the legal page, click on "Yes, I agree to the above terms".
9. Click the download button(s) next to the file name(s) of the file(s) you wish to download.

   Note: The product names are appended with one of the letters “V”, for the VCS/VCSMX version, or “M”, for the MTI/Questa version, or “N”, for the NC version.
10. Follow browser prompts to select a destination location.
11. You may download multiple files simultaneously.

2.6.2 Downloading Using FTP with a Web Browser

1. Follow the above instructions through the product version selection step
2. Click the "Download via FTP" link instead of the "Download Here" button
3. Click the "Click Here To Download" button
4. Select the file(s) that you want to download
5. Follow browser prompts to select a destination location

2.6.3 Setting Up FlexLM Licensing

The VIP can be built to run in a FlexLM licensing environment. If you use FlexLM licensing, follow the instructions below (as well as those in README.64-bit if you are using a 64-bit system.)

2.6.3.1 Downloading and Installing the DesignWare Libraries

A shared library in the Synopsys DesignWare suite is needed to interface between the VIP and the FlexLM mechanism. If you have not already installed the DesignWare library, follow these steps to do so:

1. Log into the Synopsys SolvNet Download Center at the following link:

   https://solvnet.synopsys.com/DownloadCenter/dc/product.jsp
2. Click on “VCS Verification Library” near the bottom of the list.

3. Click on the VCS version you are using, for example “F-2011.12”.
4. Click “Download Here” button
5. Click “YES, I AGREE TO THE ABOVE TERMS” to acknowledge the download terms.
6. Click on the Download button for the dw_vip_common_5.95a.run file.
7. Save the dw_vip_common_5.95a.run file to a UNIX directory.
8. From a UNIX prompt create a dw_home directory and execute the following:

   ```
   $ dw_vip_common_5.95a.run --dir path/dw_home
   ```
9. Use the following command to set the DESIGNWARE_HOME environment variable to the `path` you specified for the dw_home directory:

   ```
   $ setenv DESIGNWARE_HOME path/dw_home
   ```

2.6.3.2 Running Simulations in the FlexLM Environment

An additional command is needed to pull in the above DesignWare library. The additional command varies for the different simulators, as shown below. Where *`platform`* is specified in the command, replace it by the appropriate entry from the Platform column in Table 2-2.

Table 2-2 Platform designations for supported platform types

Platform type	Simulator	Platform
64-bit Linux	64-bit	amd64
64-bit Linux	32-bit	linux
32-bit Linux	32-bit	linux
32-bit Solaris	32-bit	solarisx86
64-bit Solaris	64-bit	x86solaris64

2.6.3.2.1 VCS/VCS MX

To include the DW common library, add the following to your simulators command line:

```
"$DESIGNWARE_HOME/vip/common/latest/C/lib/platform/VipCommonNtb.so"
```

2.6.3.2.2 NC

To include the DW common library, add the following to your simulators command line:

```
"-sv_lib $DESIGNWARE_HOME/vip/common/latest/C/lib/platform/VipCommonNtb.so"
```

2.6.3.2.3 MTI/Questa

To include the DW common library, add the following to your simulators command line:

```
"-R -sv_lib $DESIGNWARE_HOME/vip/common/latest/C/lib/platform/VipCommonNtb -"
```

Note: Be sure to include the trailing hyphen (-).

When running commands with `make`, these additional command line arguments are added with the VC_ARGS `make` variable. For example, to run the test named test_basic.fl in the Example directory, use the following command line (where the simulator is VCS on a 32-bit system):

```
% make test_basic VC_ARGS="$DESIGNWARE_HOME/vip/common/latest/C/lib/linux/ \
     VipCommonNtb.so"
```

2.6.4 32-bit and 64-bit Simulation Modes

On most simulators you can run in either a native 64-bit mode, or a legacy-emulated 32-bit mode.

Running in emulated 32-bit mode uses the same PLIs as those used on 32-bit machines. Generally they do not even need to be recompiled. This can be useful on a a multi-machine heterogeneous grid with both 32 and 64-bit processors, since a single PLI can be shared across machines. The downside is that simulations that require very large amounts of memory (greater than 3GB) will be memory-constrained by the 32-bit library.

Compilation of the 32-bit PLI typically requires special command-line switches to inform the compiler/linker to create 32-bit object files (see "Running in 32-bit mode" and "Run a 64-bit executable using 32-bit PLI object").

In the native 64-bit mode, you must use a 64-bit PLI. If a 64-bit mode compiler is available, there generally is no need for special methods to build the PLI objects.

2.6.4.1 Settings for Synopsys VCS/VCS MX

There are two basic ways to run VCS/VCS MX simulations on a 64-bit machine:

- "Run a 64-bit executable using 32-bit PLI object"
- "Run a 64-bit executable using a 64-bit PLI object:"

2.6.4.1.1 Run a 64-bit executable using 32-bit PLI object

Compile VCS in 64-bit mode, but generate a 32-bit executable.

1. Rebuild the 32-bit PLIs, after cleaning out the old ones:

   ```
   % setenv SVC_USE_64BIT_LNX 1
   % cd installationDir/Verilog
   % make clean
   % make
   ```

2. When compiling or running VCS, add the `-comp64` option (note that the `-full64` option is not used here).

2.6.4.1.2 Run a 64-bit executable using a 64-bit PLI object:

Run VCS with the `-full64` option, having it compile your PLI. The command to do this is similar to the following:

```
vcs -CFLAGS "-I$TOP/PCIE/Verilog" -sverilog -full64 -R -q +cli +plusarg_save +v2k -P
$TOP/PCIE/Verilog/Util/pli.tab
$TOP/PCIE/Verilog/Util/Msglog/msglog.c +incdir+$TOP/PCIE/Verilog
-file test.fl
```

(Note that the msglog.c file is specified in the command, not the object.)

Alternatively, use the `vcs` script to run it. For example:

```
Util/vcs -f test_basic.fl
```

2.6.4.2 Settings for Cadence NC

There are two basic methods for running NC simulations on a 64-bit machine:

- "Running in 32-bit mode", using 32-bit PLI library
- "Running in 64-bit mode" with a 64-bit PLI library

2.6.4.2.1 Running in 32-bit mode

1. Rebuild the 32-bit PLIs, after cleaning out the old ones:

   ```
   % setenv SVC_USE_64BIT_LNX 1
   % cd installationDir/Verilog
   % make clean
   % make
   ```

2. Run the simulation using the 32-bit version of NC. For example:

   ```
   % cd Example
   % make test
   ```

2.6.4.2.2 Running in 64-bit mode

Running in 64-bit mode enables a much larger available memory. PLIs compiled for 64-bit must be used with the `irun` 64-bit command.

Your search path must be set to find a `gcc` version in a *x86_64 directory. To check which version of the `gcc` tool is found in your search path, use the following command:

```
unix> gcc --version
```

After verifying that the desired version of gcc is being found, test your environment by typing the following command:

```
% gcc -dumpmachine
```

and make sure that it returns the x86_64 variant.

To keep the compiler/linker from building 32-bit libraries, make sure the environment variable SVC_USE_64BIT_LNX is unset:

```
% unsetenv SVC_USE_64BIT_LNX
```

Build the libraries and create a .so file:

```
% cd installationDir/Verilog
% make clean
% make
```

Use the following command to verify that the libraries are 64-bit:

```
% file dyn_libpli_lnx86.so
```

When you run the `file` command on the .so file created above, it should return "ELF 64-bit ...". After you have verified that, rerun the simulation with the PATH to this 64-bit .so file, and you will be able to use the .so file in a 64-bit `irun` simulation.

2.7 Building the Tools, Header Files and PLI Libraries

2.7.1 Makefiles

The VIP comes with Makefiles for various platforms, which have a number of platform-specified variations. These include basic UNIX tools (such as `ls`, `cat`, `sed`, and so on), default development tools (such as `cc`, `make`, and so on), and additional site-specific tools and scripts.

The installer script runs `make` at the *installationDir*/PCIE/Verilog directory level to create shared libraries for each platform supported: Linux, Solaris, 32-bit, and 64-bit.

After the `make` completes, verify that the dyn_libpli.so file is present in the *installationDir*/PCIE/Verilog directory.

If you need to clean up everything and start from scratch, run a `make clobber`. To just rebuild the PLI libraries, run `make clean` followed by `make` in the PCIE/Verilog directory.

2.7.2 Helper Scripts

The VIP includes sample invocation scripts that are customized for each particular simulator. The helper scripts are located in the Util directory and are created when you run `make`. The scripts are named `ncs`, `mts`, `vcs`, and `param2def`.

`vcs` – Invokes the VCS compiler/simulator with appropriate arguments.

`ncs` – Invokes the NC-Verilog compiler/simulator with appropriate arguments.

`mts` – Invokes the MTS compiler/simulator with appropriate arguments.

`param2def` – Converts a file of Verilog parameters to the equivalent C header file.

To see information about using a helper script, call it with the `-help` option. For example:

```
vcs -help
```

You must set an additional environment variable before you use any of the invocation helper scripts:

*product*SVC_INCDIRS (for example, PCIESVC_INCDIRS)

Set the variable to the include path for include dirs. The syntax is similar to the `+incdir` syntax:

```
+DIR1+DIR2+...
```

If the *product*SVC_INCDIRS variable is not set, the helper invocation script will attempt to infer the directory locations from the current (script) location, and set the variable to the top-level directory (*installationDir*/PCIE/*simulatorType*).

2.7.2.1 Running the Scripts

Change to the directory that contains the test you want to run, then execute the appropriate command below.

2.7.2.1.1 VCS/VCS MX

Run `*topLevelDirectory*/Util/vcs -f filelist`

NOTE:
See "Running Simulations in the FlexLM Environment" if your are running with FlexLM licensing.

2.7.2.1.2 NC

Run `*topLevelDirectory*/Util/ncs -f filelist`

2.7.2.1.3 MTI/Questa

Run `*topLevelDirectory*/Util/mti -f filelist`

2.7.3 Setting Up an Example

Examples are provided for all supported simulators.

To build the Verilog example in *installationDir*/PCIE/Verilog/Example, for instance, run make as follows:

```
make test_basic VC_ARGS="-V $DESIGNWARE_HOME/vip/common/latest/C/lib/amd64/
VipCommonNtb.so"
```

where the *lib* directory matches your compiler (32 or 64 bit) and OS (linux, amd64, suse32, suse64, and so on).

VC_ARGS is used to supply options to the scripts and simulator.

You can use -V to echo output to stdout, otherwise all output goes to /dev/null and only the log file will contain the output.

2.8 Running a Test

Note that your own environment may require additional libraries and defines.

Go to the directory that contains the test you want to run, and execute the command below that applies to your simulator:

The `make` and scripts provided are for illustration only, enabling the examples to run. You should review these and incorporate the appropriate commands into your environment.

2.8.1 VCS/VCS MX

```
vcs commandLineArguments [+define ...] -file filelist
```

2.8.2 NC

```
ncverilog commandLineArguments [+define ...] -f filelist
```

2.8.3 MTI/Questa

```
vlog commandLineArguments [+define ...] -f filelist
```

2.9 Licensing

The VC VIP for PCI Express uses the Synopsys Common Licensing (SCL) software to control its usage. You can find general SCL information at:

http://www.synopsys.com/keys

PCIe VIP licensing requirements vary depending on whether the Discovery VIP or the SVC VIP is being used, whether the VIP is a Gen2 or Gen3 VIP, and whether it is a Verilog or SVT VIP. Figure 2-1 shows the license search sequence for Discovery VIPs. Figure 2-2 shows the license search sequence for SVC VIPs.

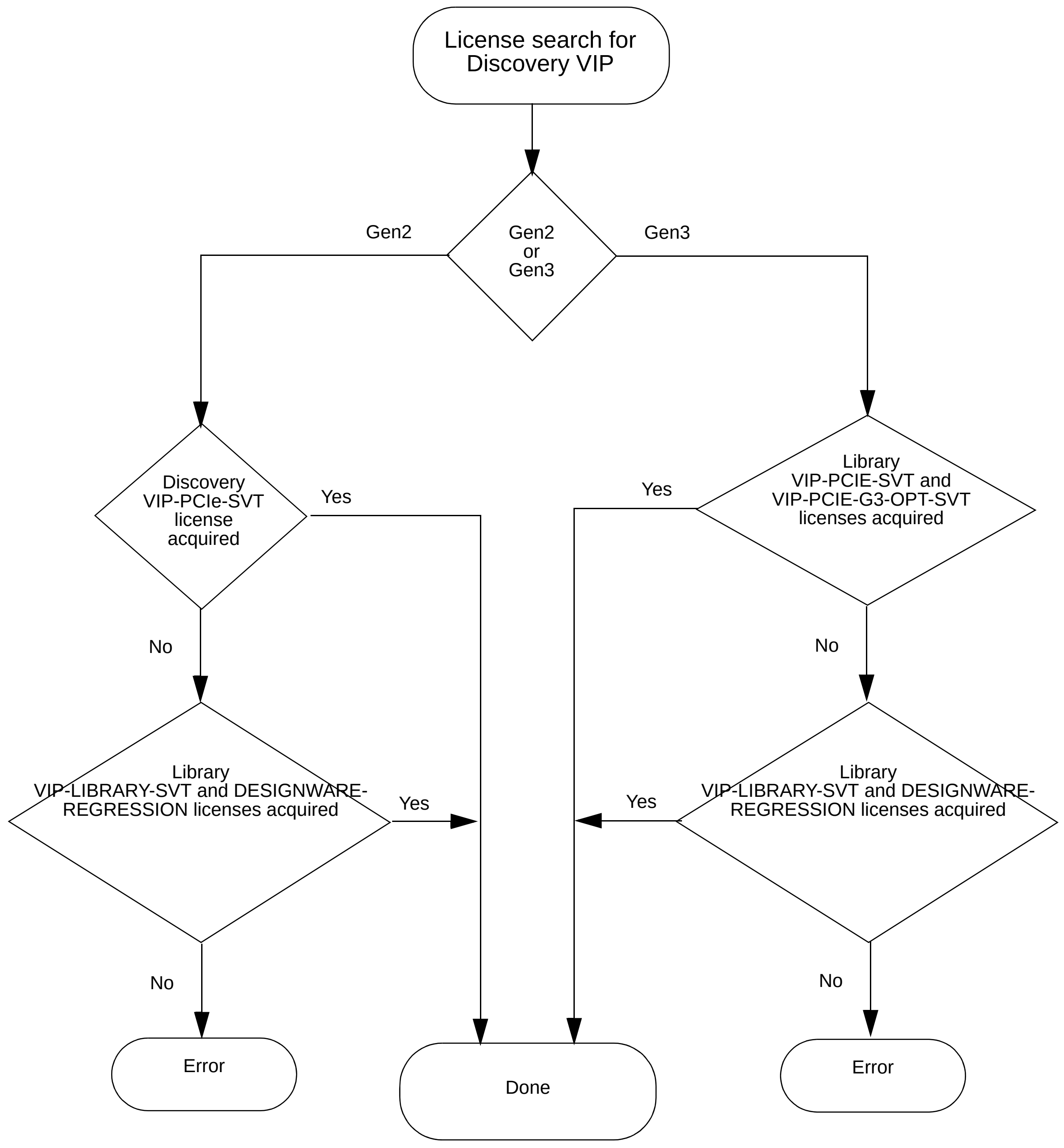

License search for Discovery VIP
Gen2 or Gen3
Gen2
Gen3
Discovery VIP-PCIe-SVT license acquired
Yes
No
Library VIP-PCIE-SVT and VIP-PCIE-G3-OPT-SVT licenses acquired
Yes
No
Library VIP-LIBRARY-SVT and DESIGNWARE-REGRESSION licenses acquired
Yes
No
Library VIP-LIBRARY-SVT and DESIGNWARE-REGRESSION licenses acquired
Yes
No
Error
Done
Error

Figure 2-1 Discovery PCIe VIP license acquisition sequences

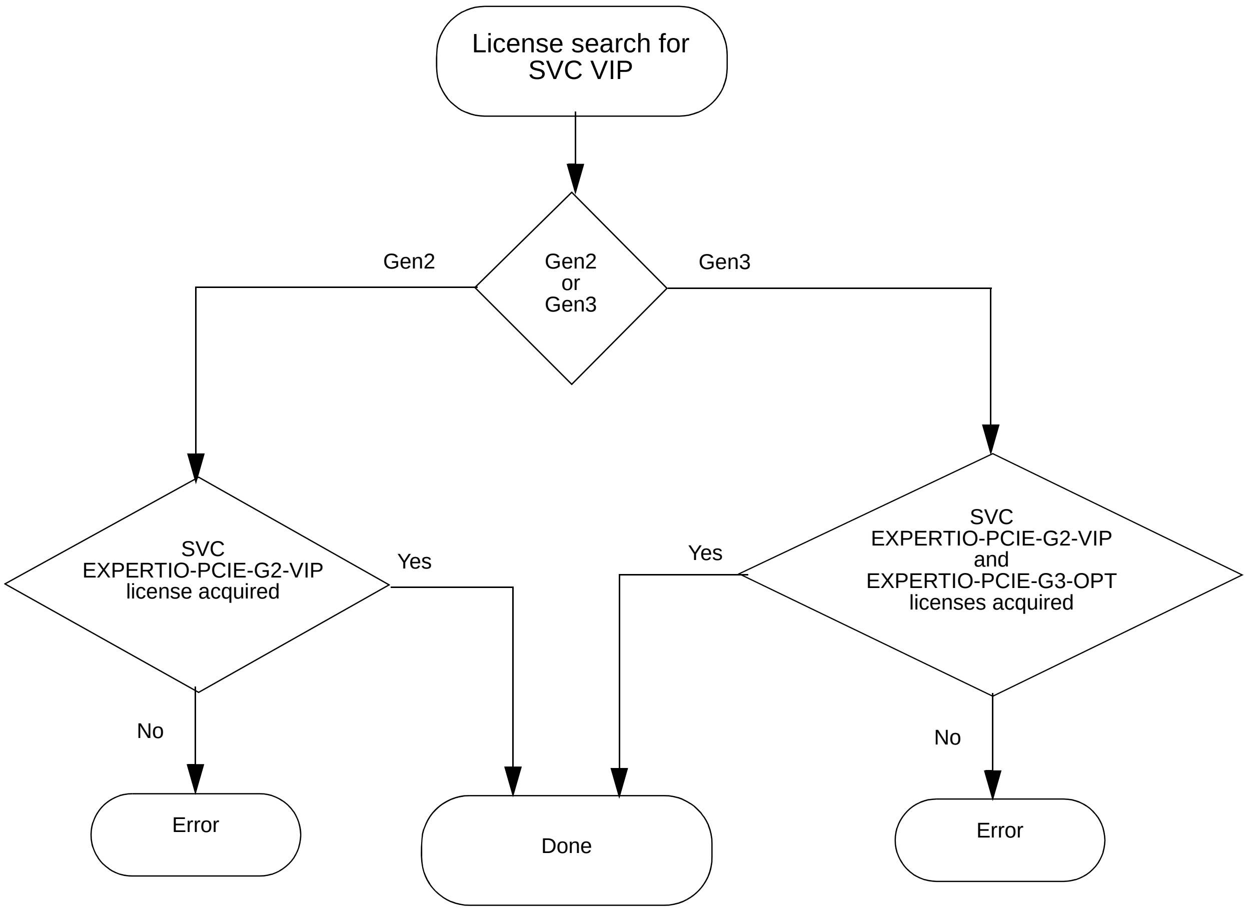

License search for SVC VIP
Gen2 or Gen3
Gen2
Gen3
SVC EXPERTIO-PCIE-G2-VIP license acquired
Yes
No
SVC EXPERTIO-PCIE-G2-VIP and EXPERTIO-PCIE-G3-OPT licenses acquired
Yes
No
Error
Done
Error

Figure 2-2 PCIe SVC VIP license acquisition sequences

Only one license is consumed per simulation session, no matter how many VIP products are instantiated in the design.

The licensing key must reside in files that are indicated by specific environment variables. For information about setting these licensing environment variables, refer to "Setting Environment Variables" on page 32.

2.10 Controlling License Usage

2.10.1 License Polling

If you request a license and none are available, license polling allows your request to exist until a license becomes available instead of exiting immediately. To control license polling, you use the DW_WAIT_LICENSE environment variable as follows:

- To enable license polling, set the DW_WAIT_LICENSE environment variable to 1.
- To disable license polling, unset the DW_WAIT_LICENSE environment variable. By default, license polling is disabled.

2.10.2 Simulation License Suspension

All VIP products support license suspension. Simulators that support license suspension allow a model to check in its license token while the simulator is suspended, then check the license token back out when the simulation is resumed.

Note: This capability is simulator-specific; not all simulators support license check-in during suspension.

2.11 Troubleshooting

This section provides troubleshooting tips if you encounter problems with licensing, installation, setting up your environment, and supported tools.

2.11.1 Licensing Problems

Q: *My simulations are exiting with an error. I suspect it is a licensing issue. How do I fix this?*

A: By default, simulations exit with an error when a license cannot be secured. Alternatively, the DW_NOAUTH_CONTINUE environment variable can be set to allow simulations to continue when one or more VIP models fail to authorize. Unauthorized VIP models essentially become disabled when DW_NOAUTH_CONTINUE is set to any value.

```
% setenv DW_NOAUTH_CONTINUE
```

Also, some simulation environments allow *license polling*, which pauses the simulation until a license is available. For more information about license polling, see “License Polling” on page 41. If you encounter further licensing issues, contact customer support using the process described in “Customer Support” on page 10.

2.11.2 Problems with Compiling or Running

- Examine the initial lines of output to determine problems. Sometimes “cascading” problems cause a single simple issue to look like multiple problems.
- Typically, problems compiling the Verilog are caused by an incorrect include search path. Make sure that either the default path will work for you or that the *product*SVC_INCDIRS environment variable is set to the correct path.
- For problems with a missing PLI library (dyn_libpli.so or plilib.so), check the following:
  - Verify that the library is where you expect it to be.
  - Check the setting of the LD_LIBRARY_PATH environment variable.
  - Did you run make at the beginning to build the object files and shared libraries?
  - Make sure the permissions on the library are set so that it is readable by you.
  - Did you compile a different type (such as 64-bit) PLI than you are simulating with?
- For compiler problems, make sure that the compiler is not an old K&R compiler because those compilers are not ANSI compatible. Use an ANSI-compatible compiler.
- For runtime errors reported in encrypted segments of VIP models, check that all the necessary +define statements are provided and that the paths are correct.

2.12 Downloading Protocol Analyzer

Protocol Analyzer is a separate download from the PCIe VIP. To find the latest version, login to SolvNet and then perform the following steps:

1. Select Downloads from the top menu bar.
2. Select the Complete Product List tab.
3. Select view products for A-G.
4. Under A-G > view products, select Discovery Verification IP.
5. Select the version you want to download. .
6. Select one of these three options:

   Download Here | via FTP | FTP Download Instructions
7. Click the YES, I AGREE TO THE ABOVE TERMS button to accept the download agreement terms.
8. Under "File" in the table, look for "vip_pa*", where "*" represents a version, and click the Download button for the version you want to download.

3 Applications

The software application layer is responsible for generating and handling transactions. To accelerate testcase development, Synopsys provides a *driver* for creating transaction requests and tracking completions as well as a *completion target* for handling the transactions and returning completions.

3.1 Driver Application

The driver provides a simple interface for users to quickly create transactions and track the corresponding completions. Transactions are created by calling tasks, though some TLP fields will be set through configuring _VARs discussed in the next section.

3.1.1 Driver Parameters

Driver parameters are listed in Table 3-1.

Table 3-1 Driver parameters

Parameter Name	Settable VAR?*	Type	Range	Default	Description
PCIE_SPEC_VER					
	Yes	Real	1.1, 2.0, 2.1, 3.0	PCIE_SPEC_VER_2_1	See Include/pciesvc_parms.v: PCIE_SPEC_VER_* Note: Please set this in the model.
PERCENTAGE_USE_TLP_DIGEST					
	Yes	Integer	0-100	0	Percentage probability of the TD bit being set, indicating that the packet has a TLP digest.
SET_NO_SNOOP_FIELD					
	Yes	Integer	0-1	0	Sets the value of the no snoop bit on all outgoing TLPs, when applicable.

Table 3-1 Driver parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
SET_IDO_FIELD					
	Yes	Integer	0-1	0	Sets the value of the IDO bit in all TLPs, if applicable.
SET_RELAXED_ORDERING_FIELD					
	Yes	Integer	0-1	0	Sets the value of the relaxed ordering bit in all TLPs, if applicable for that TLP type.
MIN_TIME_TO_NEXT_TRANSACTION					
	Yes	Integer	0-large value	0	Minimum time between transactions before the driver will transmit another queued request.
MAX_TIME_TO_NEXT_TRANSACTION					
	Yes	Integer	0-large value	0	Maximum time between transactions in before the driver will transmit another queued request.
DISPLAY_OUTSTANDING_COMMANDS_PERIOD					
	Yes	Integer	10_000-large value	50_000	Driver will display all outstanding/pending commands periodically according to this parameter.
READ_CPL_BOUNDARY					
	Yes	Integer	64, 128	64	Driver will check that all read completions that do not terminate the transaction have a data payload in bytes that is modulo of this value.
COMPLETION_TIMEOUT_NS					
	Yes	Integer	10,000-large value	500_000	If the request is not completed by the time COMPLETION_TIMEOUT has elapsed, the request has timed out and the driver will issue a warning.
MAX_NUM_TAGS					
	No	Integer	1-256	32	Maximum number of tags that can be used. If greater than 32 it is assumed that the extended tag bits are legal to use.
DISCARD_COMPLETIONS					
	Yes	Integer	0-1	0	When set to a 1, the driver will immediately upon reception of a completion discard the status of completed commands. Completion status cannot be checked in this mode. Users should only set this to 1 if they will not be checking completion results.

Table 3-1 Driver parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
CMB_TABLE_SIZE					
	No	Integer	16-256	256	Size of the command management block, which is used to track pending and outstanding transactions.
ENABLE_TX_TLP_REPORTING					
	Yes	Integer	0-1	0	If enabled, transmitted TLPs will be reported to the global shadow memory. This should only be turned on if the corresponding LinkLayer parameter is not turned on (or the VIP LinkLayer doesn't exist, as when using the driver to driver a DUT.) Make sure this is not enabled if the shadow memory is not instantiated.
ENABLE_SHADOW_MEMORY_CHECKING					
	Yes	Integer	0-1	1	When set to 1, the returned read transactions are checked against their expected value in the Global Shadow Memory. If the Global Shadow is not being used, this should be set to 0.
DISPLAY_NAME					
	No	String		"pciesvc_driver"	Default display name for the driver.
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with "_VAR". This is useful if using higher level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.					

3.1.2 Driver User Tasks

The tasks listed in Table 3-2 may be called to generate transactions and check for completion status. Certain bits that are less frequently user or changed must be set using the variables mentioned in the previous section.

Note that data used with the QueueCfgWrite and QueueCfgRead tasks is byte swapped from the received TLP format. All other transaction types are endian neutral and use the same format as contained in the TLP.

Table 3-2 Driver user tasks

Task Name	Arguments	I/O	Values
SetRequesterID Sets the requester ID that the driver will use for all queued transactions. This will also build and populate the configuration block for this particular driver instance.	req_id(16)	I	Requester ID that the driver is supposed to use. This ID is a concatenation of the bus number, device number and function number.
QueueMemRead Queue a memory read TLP request for transmission. Memory read transactions are non-posted and require a completion.	address(64)	I	Destination address.
	length(32)	I	Length of the data read request
	first_dw_be(4)	I	First data word byte enable
	last_dw_be(4)	I	Last data word byte enable
	traffic_class(32)	I	Traffic class of the request
	address_translation(2)	I	Address translation field. See PCIE base spec for more details.
	buffer_ptr(32)	I	Buffer that the read data should be written to when the transaction is completed.
	buffer_len(32)	I	Buffer size in 32bit dwords
	error_injection_code(32)	I	Error injection code.*
	block(1)	I	Indication on whether or not task will block until a completion is returned.
	command_num (32)	O	Unique command number for this transaction.
	completion_status(32)	O	Completion status of transaction. Valid only if blocking.**

Table 3-2 Driver user tasks (Continued)

Task Name	Arguments	I/O	Values
QueueMemReadLocked Queue a memory read locked TLP request for transmission. Memory read transactions are non-posted and require a completion.	address(64)	I	Address to be read
	length(32)	I	Length of the data read request
	first_dw_be(4)	I	First data word byte enable
	last_dw_be(4)	I	Last data word byte enable
	traffic_class(32)	I	Traffic class of the request
	address_translation(2)	I	Address translation field. See PCIE base spec for more details.
	buffer_ptr(32)	I	Buffer that the read data should be written to when the transaction is completed.
	buffer_len(32)	I	Buffer size in 32bit dwords
	error_injection_code(32)	I	Error injection code.*
	block(1)	I	Indication on whether or not task will block until a completion is returned.
	command_num(32)	O	Unique command number assigned to this transaction.
	completion_status(32)	O	Completion status of transaction. Valid only if blocking.**
QueueMemWrite Queue a memory write TLP request for transmission. Memory write transactions are posted and do not require a completion.	address(64)	I	Memory write destination address.
	length(32)		Length field for the transaction.
	first_dw_be(4)	I	First data word byte enables.
	last_dw_be(4)	I	Last data word byte enables.
	traffic_class(32)	I	Traffic class of the request.
	address_translation(2)	I	Address translation field. See PCIE base spec for more details.
	ep(1)	I	A ‘1’ sets the error poison bit.
	buffer_ptr(32)	I	Pointer to the data to be written
	buffer_len(32)	I	Buffer size in 32 bit dwords
	error_injection_code(32)	I	Error injection code.*
	block(1)	I	Indication on whether or not task will block until transaction is complete. For posted commands such as MemWrite the command is considered complete as soon as the driver has sent it.
	command_num(32)	O	Unique command number assigned to this transaction.
	completion_status(32)	O	Completion status of the transaction. Valid only if blocking.**

Table 3-2 Driver user tasks (Continued)

Task Name	Arguments	I/O	Values
QueueIORead Queue an I/O read TLP request for transmission. I/O transactions are non-posted and require a completion.	io_address(32)	I	Destination address.
	first_dw_be(4)	I	First data word byte enable
	buffer_ptr(32)	I	Pointer to the data to be written.
	buffer_len(32)	I	Buffer size in 32 bit dwords
	error_injection_code(32)	I	Error injection code.*
	block(1)	I	Indication on whether or not task will block until a completion is returned.
	command_num(32)	O	Unique command number assigned to this transaction.
	completion_status(32)	O	Completion status of the transaction. Valid only if blocking.**
QueueIOWrite Queue an I/O write TLP request for transmission. I/O transactions are non-posted and require a completion.	io_address(32)	I	Destination address.
	data(32)	I	Dword of data to be written
	first_dw_be(4)	I	First data word byte enable
	ep(1)	I	Sets the EP bit in the TLP indicating that the data is poisoned.
	error_injection_code(32)	I	Error injection code.*
	block(1)	I	Indication on whether or not task will block until a completion is returned.
	command_num(32)	O	Unique command number assigned to this transaction.
	completion_status(32)	O	Completion status of transaction. Only valid if blocking.**

Table 3-2 Driver user tasks (Continued)

Task Name	Arguments	I/O	Values
QueueCfgRead Queue a type 0 or type 1 Config read request, based on the config_type argument, for transmission. Configuration requests are non-posted and require a completion. Data used with the QueueCfgRead task is byte swapped from the received TLP format. This is done for the ease of use by the customer since by definition PCIE Configuration Space is little endian.	cfg_address(16)	I	Destination address to send the Config read to. Note that configuration requests route by ID. Also known as {Bus, Device, Function Number}.
	register_number(10)	I	Configuration reg number to be read. The lower 6 bits are the register number and the upper 4 bits are the extended register number.
	first_dw_be(4)	I	First data word byte enable
	config_type(1)	I	0 means type 0 read and 1 means type 1 read.
	buffer_ptr(32)	I	Pointer to the data to be written. Note the data returned will be in little endian format like the configuration space (i.e. byte swapped from the TLP ordering)
	buffer_len(32)	I	Buffer size in 32 bit dwords
	error_injection_code(32)	I	Error injection code.*
	block(1)	I	Indication on whether or not task will block until a completion is returned.
	command_num(32)	O	Unique command number assigned to this transaction.
	completion_status(32)	O	Completion status of transaction. Only valid if blocking.**
QueueCfgWrite Queue a type 0 or type 1 config write request, based on the config_type argument, for transmission. Configuration requests are non-posted and require a completion. Data used with the QueueCfgWrite task is byte swapped from the received TLP format. This is done for the ease of use by the customer since by definition PCIE Configuration Space is little endian.	cfg_address(16)	I	Destination address to be written to. Note that configuration requests route by ID. This address will be captured as the Completer ID in the endpoint. Also known as {Bus, Device, Function Number}.
	register_number(10)	I	Configuration register number to be read. The lower 6 bits are the register number and the upper 4 bits are the extended register number.
	first_dw_be(4)	I	First data word byte enable
	config_type(1)	I	0 means type 0 config write. 1 means type 1 config write.
	ep(1)	I	Error poison bit. Indication that the data in the payload is corrupt.
	write_data(32)	I	Dword of data to be written in little Endian format (same as configuration space, and byte swapped from TLP ordering)
	error_injection_code(32)	I	Error injection code for this transaction.*
	block(1)	I	Indication on whether or not task will block until a completion is returned.
	command_num(32)	O	Unique command number assigned to this transaction.
	completion_status(32)	O	Completion status of transaction. Only valid if blocking.**

Table 3-2 Driver user tasks (Continued)

Task Name	Arguments	I/O	Values
QueueMsg Queue a Msg TLP if no payload is present or a MsgD command if a payload is present. Message requests are posted and do not require a completion.	vendor_fields(64)	I	Bytes 8-12 of the TLP header are for a vendor defined message. This should be set to 0 if a message other than a vendor defined message is being sent.
	traffic_class(32)	I	Traffic class of message to be sent.
	message_code(8)	I	Message code to be sent.
	routing_type(3)	I	Message routing type.
	ep(1)	I	Indication that data payload is corrupt. Should always be 0 unless the message contains a payload.
	buffer_ptr(32)	I	Pointer to message payload, if any. Otherwise this field is reserved and should be set to 0.
	buffer_length(32)	I	Length of the data payload in dwords. Payloads should be present only for vendor defined messages. Length must be set to 0 if no payload is present.
	error_injection_code(32)	I	Error injection code for this TLP.*
	block(1)	I	Indication on whether or not task will block until transaction is complete. For posted requests such as this one, the transaction is complete once the driver has transmitted it.
	command_num(32)	O	Unique command number assigned to this transaction.
	completion_status(32)	O	Status of the command request.**

Table 3-2 Driver user tasks (Continued)

Task Name	Arguments	I/O	Values
QueueFetchAdd Queue a FetchAdd Atomic Operation TLP request for transmission. FetchAdd transactions are non-posted and require a completion.	address(64)	I	Destination address.
	length(32)	I	Length of the data read request
	first_dw_be(4)	I	First data word byte enable
	last_dw_be(4)	I	Last data word byte enable
	traffic_class(32)	I	Traffic class of the request
	address_translation(2)	I	Address translation field. See PCIE base spec for more details.
	ep(1)	I	Error poison bit. Indication that the data in the payload is corrupt.
	data_in_buffer_ptr(32)	I	Buffer that stores the completion data coming in.
	data_in_buffer_len(32)	I	Buffer size in 32bit dwords
	data_out_buffer_ptr(32)	I	Buffer that stores the outgoing data payload
	data_out_buffer_len(32)		Buffer size in 32bit dwords
	error_injection_code(32)	I	Error injection code*
	block(1)	I	Indication on whether or not task will block until a completion is returned.
	command_num (32)	O	Unique command number for this transaction.
	completion_status(32)	O	Completion status of transaction. Valid only if blocking.**

Table 3-2 Driver user tasks (Continued)

Task Name	Arguments	I/O	Values
QueueSwap Queue a Swap Atomic Operation TLP request for transmission. Swap transactions are non-posted and require a completion.	address(64)	I	Destination address.
	length(32)	I	Length of the data read request
	first_dw_be(4)	I	First data word byte enable
	last_dw_be(4)	I	Last data word byte enable
	traffic_class(32)	I	Traffic class of the request
	address_translation(2)	I	Address translation field. See PCIE base spec for more details.
	ep(1)	I	Error poison bit. Indication that the data in the payload is corrupt.
	data_in_buffer_ptr(32)	I	Buffer that stores the completion data coming in.
	data_in_buffer_len(32)	I	Buffer size in 32bit dwords
	data_out_buffer_ptr(32)	I	Buffer that stores the outgoing data payload
	data_out_buffer_len(32)		Buffer size in 32bit dwords
	error_injection_code(32)	I	Error injection code*
	block(1)	I	Indication on whether or not task will block until a completion is returned.
	command_num (32)	O	Unique command number for this transaction.
	completion_status(32)	O	Completion status of transaction. Valid only if blocking.**

Table 3-2 Driver user tasks (Continued)

Task Name	Arguments	I/O	Values
QueueCAS Queue a CAS (compare and swap) Atomic Operation TLP request for transmission. CAS transactions are non-posted and require a completion.	address(64)	I	Destination address.
	length(32)	I	Length of the data read request
	first_dw_be(4)	I	First data word byte enable
	last_dw_be(4)	I	Last data word byte enable
	traffic_class(32)	I	Traffic class of the request
	address_translation(2)	I	Address translation field. See PCIE base spec for more details.
	ep(1)	I	Error poison bit. Indication that the data in the payload is corrupt.
	data_in_buffer_ptr(32)	I	Buffer that stores the completion data coming in.
	data_in_buffer_len(32)	I	Buffer size in 32bit dwords
	data_out_buffer_ptr(32)	I	Buffer that stores the outgoing data payload
	data_out_buffer_len(32)		Buffer size in 32bit dwords
	error_injection_code(32)	I	Error injection code*
	block(1)	I	Indication on whether or not task will block until a completion is returned.
	command_num (32)	O	Unique command number for this transaction.
	completion_status(32)	O	Completion status of transaction. Valid only if blocking.**
IsTransactionComplete Checks if the transaction specified by request_id is complete. When this task returns a positive status, the result of the command is purged from driver memory. Any repeat calls to this task with the same command number will return a status of unknown because the command result is no longer in the driver memory.	command_num(32)	I	Unique command number assigned to this transaction.
	completion_status(32)	O	Status of the transaction. Valid only if the command is complete.**
	result(1)	O	If 1 then the command is complete. If 0 then the command has not yet completed.
AddTLPPrefix Adds the prefix or prefixes contained in prefix_array to the next queued command.	logic [31:0] prefix_array[]	I	A dynamic array containing the prefixes to be added. It is up to the user to build and set the contents of the array.

Table 3-2 Driver user tasks (Continued)

Task Name	Arguments	I/O	Values
WaitForCompletion Blocks until the transaction specified by request_id has completed. The completion status is purged from driver memory after this task finished.	command_num(32)	I	Unique command number assigned to this transaction.
	completion_status(32)	O	Final completion status.**
WaitUntilDriverIdle Blocking task that finishes when all outstanding requests have been completed and there are no queued requests waiting to go out.	n/a	n/a	
SetUseOrderedTags When set the driver will issue tags in ascending numerical order rather than picking a random value	n/a	n/a	

* -These parameters for this field start with TX_TLP_EI_* and are defined in Verilog/Include/pciesvc_parms.v. The error_injection_code field here is intended to be used for directed tests. For random error injection it is recommended to utilize the facilities defined in the Data Link Layer.

**- The parameters for completion_status start with TLP_CPL_STATUS_* and can be found in Verilog/Include/pciesvc_tlp_parms.v

3.1.3 Driver Memory Management Requirements

Most of the QueueXYZ driver tasks require the user to supply some sort of memory buffer. The driver will not deallocate these memory buffers since they were allocated externally. The type of command will determine when a user may deallocate or reuse a buffer that was passed to the driver.

Posted commands such as config writes and memory write do not require the memory buffer at all after the QueueXYZ command has been called. Thus if QueueMemWrite() was called 10 times consecutively, a user could simply use the same memory pointer and change the data for each call.

A memory pointer used for non-posted commands must not be deallocated until the command has completed or timed out. There are several ways to find out if a command is complete:

- Use the blocking argument when queuing the command
- Call WaitForCompletion() to wait until a specific command has completed
- Call IsTransactionComplete() to find out if a queued command has completed.
- Call WaitUntilDriverIdle to wait until all commands have completed.

Once the command is complete, the memory buffer may if desired be read for completion data and then should be deallocated.

Failure to deallocate unused memory may result in the simulation crashing due to a lack of VIP memory in test cases with a large number of transactions.

3.1.4 Driver Events

3.1.4.1 event_transaction_completed

This event will trigger after a transaction has been completed by receiving successful completion status with the byte count for the transaction exhausted, a completion status of unsupported request, completer abort or completion retry status, or if a command times out .

3.2 Request Initiator Application

The Requester application generates PCIE Read/Write transactions to a target. Memory addresses are chosen at random (constrained by min/max parameters) and have varying lengths (again, constrained by min/max parameters.)

3.2.1 Requester Parameters

Requester parameters are listed in Table 3-3.

Table 3-3 Requester parameters

Parameter Name	Settable VAR?*	Type	Range	Default	Description
NUM_MEM_WRITE					
	Yes	Integer	0-large value	20	Number of memory write transactions to send.
NUM_MEM_READ					
	Yes	Integer	0-large value	20	Number of memory read transactions to send.
MIN_DATA_LEN_IN_BYTES					
	Yes	Integer	1-4096	1 bytes	Minimum/maximum data length of write and read requests.
MAX_DATA_LEN_IN_BYTES					
	Yes	Integer	0-4096	128 bytes	
BANDWIDTH_MODE					
	Yes	Integer	BANDWIDTH_MODE_RPS, BANDWIDTH_MODE_REQ_DELAY	BANDWIDTH_MODE_REQ_DELAY	Set the mechanism used to meter out the transactions. The modes available are REQ_DELAY (with a specified time between packets) and RPS (specified requests-per-second.)
MIN_TIME_BETWEEN_PACKETS_NS					

Table 3-3 Requester parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
	Yes	Integer	0-large value	1 ns	Minimum/maximum delay between each transmitted packet. After each packet transmission, this value is randomized.
MAX_TIME_BETWEEN_PACKETS_NS					
	Yes	Integer	0-large value	10_000 ns	
MIN_TLP_TRAFFIC_CLASS					
	Yes	Integer	0-7	0	Minimum randomized TC value in transmitted TLPs.
MAX_TLP_TRAFFIC_CLASS					
	Yes	Integer	0-7	0	Maximum randomized TC value in transmitted TLPs.
DEFAULT_REQUESTER_ID					
	Yes	Integer	0-2^16	INVALID_ REQUESTER_ID	Default requester-ID, if no configuration requests have been seen yet. User must explicitly set this.
PERCENTAGE_USE_TLP_DIGEST					
	Yes	Integer	0-100	50	Percentage probability of the TD bit being set in a completion, indicating that the packet has a TLP digest.
MAX_NUM_TAGS					
	No	Integer	1-256	32	Maximum number of outstanding tags. Note that this must be smaller than the command management block table size (see CMB_TABLE_SIZE parameter below.)
NUM_MEM_ADDR_SEGMENTS					
	No	Integer	16-4096	10	Number of unique randomization segments – each of which is a min/max memory address range.
DISCARD_COMPLETIONS					
	Yes	Integer	0-1	0	When set to a 1, the driver will automatically discard the status of completed commands. Users should only set this parameter to a 1 if they will not be checking completion results. This parameter does not automatically deallocate buffer memories that contain data payloads. It only affects how the driver saves completion status.
CMB_TABLE_SIZE					

Table 3-3 Requester parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
	No	Integer	16-256	64 entries	Size of the command management block table, which is used to track pending and outstanding transactions.
ENABLE_TX_TLP_REPORTING					
	Yes	Integer	0-1	0	If enabled, transmitted TLPs will be reported to the global shadow memory. This should only be turned on if the corresponding LinkLayer parameter is not turned on (or the VIP LinkLayer doesn't exist, as when using the requester to driver a DUT.) Make sure this is not enabled if the shadow memory is not instantiated.
ENABLE_SHADOW_MEMORY_CHECKING					
	Yes	Integer	0-1	1	When set to 1, the returned read transactions are checked against their expected value in the Global Shadow Memory. If the Global Shadow is not being used, this should be set to 0.
DISPLAY_NAME					
	No	String		pciesvc_ requester	Default prefix in $msglog calls.
BANDWIDTH_REQUESTS_PER_SECOND					
	Yes	Integer		100,000	The variable is applicable when bandwidth_mode is set to BANDWIDTH_MODE_REQUESTS_ PER_SEC. The value of this variable specifies number of requests to be generated per second.
DISPLAY_CURRENT_STATUS_PERIOD					
	Yes	Integer		100,000	The display_current_status_period variable represents the frequency in terms of time units after which the Requester application will display its current status periodically.
DISPLAY_TLP_BEGINNING_DWORDS					
	Yes	Integer		8	Enables to display TLP beginning DWORDs.)

Table 3-3 Requester parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
DISPLAY_TLP_ENDING_DWORDS					
	Yes	Integer		4	Enables to display TLP ending DWORDs
MAX_MRD_TIMEOUT_RETRY_COUNT					
	Yes	Integer		1	The max_mrd_timeout_retry_count variable represents the maximum number of times a Memory Read request is retried before a request is determined as a failure by Requester application.
COMPLETION_TIMEOUT_NS					
	Yes	Integer		1_000_000	Completion timeout in NS .
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with “_VAR”. This is useful if using higher-level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.					

3.2.2 Requester User Tasks

The tasks listed in Table 3-4 may be called to generate transactions and check for completion status. Certain bits that are less frequently user or changed must be set using the variables mentioned in the previous section.

Table 3-4 Requester user tasks

Task Name	Arguments (length)	I/O	Values
SetRequesterId Sets the requester ID that the requester will use for all queued transactions. This will also build and populate the configuration block for this particular requester instance.	req_id (16)	I	Requester ID that the requester shall use. This ID is a concatenation of the bus number, device number and function number.
AddMemRange Adds a min/max memory address range to specify the requester-generated addresses. This task can be called additional times to add multiple ranges.	min_addr (64)	I	Minimum address to generate.
	max_addr (64)	I	Maximum address to generate.
	status (32)	O	Non-zero if AddMemRange failed.
RemoveMemRange Removes a min/max memory address range created with above AddMemRange() task. This task must be called with identical min/max memory addresses as the corresponding range you want to delete.	min_addr (64)	I	Minimum address of range to delete.
	max_addr (64)	I	Maximum address of range to delete.
	status (32)	O	Non-zero if RemoveMemRange failed.

Table 3-4 Requester user tasks (Continued)

Task Name	Arguments (length)	I/O	Values
DisplayMemRangeList Displays a list of all the memory address ranges added with AddMemRange() task, above.	n/a	n/a	
StartApp Start the Requester application.	n/a	n/a	
IsAppFinished Call to determine if the Requester Application has completed yet.	is_finished (1)	O	Set when the requester has completed.
WaitUntilFinished Blocking task to await completion of the Requester Application.	n/a	n/a	
ClearStats Clears internal statistics registers	n/a	n/a	
DisplayStats Displays the internal statistics registers.	n/a	n/a	

3.2.3 Requester Statistics

The statistics listed in Table 3-5 are maintained by the Requester:

Table 3-5 Requested statistics

Stat Name	Width	Description
stat_num_tlp_sent	32	Number of TLPs sent by the requester
stat_num_tlp_received	32	Number of TLPs received by the requester
stat_num_req_sent	32	Number of requests sent
stat_num_bytes_sent	32	Number of total bytes sent
stat_num_data_bytes_sent	32	Number of data bytes sent
stat_num_mem_req_sent	32	Number of memory requests sent
stat_num_mem_write_req_sent	32	Number of memory write requests sent
stat_num_mem_read_req_sent	32	Number of memory read requests sent
stat_num_mem_read_xn_completed	32	Number of memory read transactions completed

Table 3-5 Requested statistics (Continued)

Stat Name	Width	Description
stat_num_mem_cpl_received	32	Number of memory completions received
stat_num_bytes_received	32	Number of total bytes received
stat_num_data_bytes_received	32	Number of data bytes received

3.3 Completion Target Application

The Completion Target is a module that provides the completer services of a PCI Express completer that will respond to various inbound requests. Read transactions are optionally broken up into multiple completions, potentially interleaved with other read completions.

3.3.1 Completion Target Parameters

Completion target parameters are listed in Table 3-6.

Table 3-6 Completion target parameters

Parameter Name	Settable VAR?*	Type	Range	Default	Description
PCIE_SPEC_VER					
	Yes	Real	1.1, 2.0, 2.1, 3.0	PCIE_SPEC_VER_2_1	See Include/pciesvc_parms.v: PCIE_SPEC_VER_* Note: Please set this in the model.
DEFAULT_COMPLETER_ID					
	Yes	Integer	0-2^{16}	16'h0100	Default completer-ID, if no configuration requests have been seen yet. Note that this may be overridden both by received CfgWr requests as well as the SetCompleterID task.
MIN_READ_CPL_DATA_SIZE_IN_BYTES					
	Yes	Integer	0-4096	1 byte	Randomized [min-max] at every completion. Any read request larger than this (taking into account the Read Completion Boundary) will be broken up into multiple read completions.
MAX_READ_CPL_DATA_SIZE_IN_BYTES					
	Yes	Integer	0-4096	128 bytes	
READ_CPL_BOUNDARY					
	Yes	Integer	64, 128	64 bytes	Memory completions (except potentially first and/or last completion) will align to this boundary.
MAX_PAYLOAD_SIZE_IN_BYTES					
	Yes	Integer	2^x x=7-12	128 bytes	The TLP payload cannot exceed this value.

Table 3-6 Completion target parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
MIN_MEM_CPL_LATENCY_NS					
	Yes	Integer	0-large value	0 ns	Min/Max latency for memory transaction completions (in nanoseconds.) Note that each of multiple completions will have their own independent delays.
MAX_MEM_CPL_LATENCY_NS					
	Yes	Integer	0-large value	0 ns	
MIN_CFG_CPL_LATENCY_NS					
	Yes	Integer	0-large value	0 ns	Min/Max latency for configuration transaction completions (in nanoseconds.)
MAX_CFG_CPL_LATENCY_NS					
	Yes	Integer	0-large value	0 ns	
MIN_IO_CPL_LATENCY_NS					
	Yes	Integer	0-large value	0 ns	Min/Max latency for I/O transaction completions (in nanoseconds.)
MAX_IO_CPL_LATENCY_NS					
	Yes	Integer	0-large value	0 ns	
MIN_MSG_CPL_LATENCY_NS					
	Yes	Integer	0-large value	0 ns	Min/Max latency for message transaction completions (in nanoseconds.)
MAX_MSG_CPL_LATENCY_NS					
	Yes	Integer	0-large value	0 ns	
PERCENTAGE_USE_TLP_DIGEST					
	Yes	Integer	0-100	50	Percentage probability of the TD bit being set in a completion, indicating that the packet has a TLP digest.

Table 3-6 Completion target parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
PERCENTAGE_CORRUPT_TLP_DIGEST					
	Yes	Integer		0	Percentage of outbound transactions with a TLP digest (ECRC), that have a corrupt TLP digest (ECRC). Expected Response: DUT should drop the TLP and verification of the generation and transmission of the ERR_MSG is left up to the user.
ALLOW_VENDOR_PREFIX_L0					
	Yes	Integer		0	When set to zero the app will flag any local vendor prefix 0 that is received and treat the underlying TLP as malformed. When set to a nonzero value the target app will not flag the prefix and process the underlying TLP normally.
ALLOW_VENDOR_PREFIX_L1					
	Yes	Integer		0	When set to zero the app will flag any local vendor prefix 1 that is received and treat the underlying TLP as malformed. When set to a nonzero value the target app will not flag the prefix and process the underlying TLP normally.
ALLOW_VENDOR_PREFIX_E0					
	Yes	Integer		0	When set to zero the app will flag any end-end vendor prefix 0 that is received treat the underlying TLP as malformed. When set to a nonzero value the target app will not flag the prefix and process the underlying TLP normally.
ALLOW_VENDOR_PREFIX_E1					
	Yes	Integer		0	When set to zer,o the app will flag any end-end vendor prefix 1 that is received and treat the underlying TLP as malformed. When set to a nonzero value the target app will not flag the prefix and process the underlying TLP normally.
MAX_NUM_END_END_PREFIX_SUPPORTED					
	Yes	Integer		4	If the target app receives a TLP with more than this number of end-end TLP prefixes, the TLP will be treated as an unsupported.request.
MEM_WRITE_NOTIFICATION_MODE					
	Yes	Integer	0-1	0	When set, any received memory writes are saved in an internal queue for later examination.

Table 3-6 Completion target parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
UNINITIALIZED_MEM_READ_MODE					
	Yes	Integer	See Description	MODE_ BAAD	Controls the returned data from a read of uninitialized memory. There are four different modes (all prefixed by UNINITIALIZED_MEM_READ_MODE_): Mode — Returns MODE_ZERO — 0 MODE_BAAD — 32'hbaadbaad MODE_RANDOM — Random data MODE_CA — Completer Abort MODE_UR — Unsupported Req
FLAG_UNINITIALIZED_MEM_READ					
	Yes	Integer	0-1	1	When set, all reads to uninitialized memory will be flagged with a message. When clear, no message will come out. This does not affect the return status of the read; however, it is controlled by the above UNINITIALIZED_MEM_READ_ MODE parameter.
MEM_WRITE_NOTIFICATION_FIFO_SIZE					
	No	Integer	16-4096	64 entries	Number of queued memory-write notifications that can be queued up.
CMB_TABLE_SIZE					
	No	Integer	16-256	64 entries	Size of the command management block table, which is used to track pending and outstanding transactions.
DISPLAY_NAME					
	No	String		"pciesvc_ target"	Default prefix in $msglog calls.
APPL_ID					
	Yes	32-Bit vector		None	Application ID
CFG_WRITE_NOTIFICATION_MODE					
	Yes	Bit		1'b0	When set to 1'b1, inbound configuration write requests are saved in an internal queue for later examination. When set to 1'b0, inbound configuration write requests are not saved in an internal queue.

Table 3-6 Completion target parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
DISPLAY_CURRENT_STATUS_PERIOD					
	Yes	Bit		1'b0	When set to 1'b1, inbound configuration write requests are saved in an internal queue for later examination. When set to 1'b0, inbound configuration write requests are not saved in an internal queue.
ENABLE_64_BIT_ADDR					
	Yes	Bit		1'b1	When set to 1'b1, enables Target application to respond to 64 bit addressing of memory requests. When set to 1'b0, Target applications respond to 64 bit addressing of memory requests with CA status.
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with "_VAR". This is useful if using higher-level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.					

3.3.1.1 Completion Target User Tasks

The tasks in Table 3-7 may be called to generate transactions and check for completion status. Certain bits that are less frequently user or changed must be set using the completion target parameters listed in in Table 3-6.

Table 3-7 Completion target user tasks

Task Name	Arguments (length)	I/O	Values
SetCompleterId The default Completer ID is generally set inline via Configuration Write Requests received from the root. If the Target is to initiate any Requests before a Configuration Request is received, this task can be called to set the Completer ID. Any received Configuration Write Request will overwrite the current value.	completer_id (16)	I	Sets the default Completer ID of the Target Application. This ID consists of the {Bus, Device, Function Number}.
AddCompleterId In addition to the above "default" completer ID, this task allows configuration of additional completer IDs that this target is expected to see. It will create Cfg blocks that can be sent separate Cfg transactions.	completer_id (16)	I	Adds an additional Completer ID to the Target Application. This ID consists of the {Bus, Device, Function Number}. Note that the Bus# should match the default Completer ID's bus# (above) – as a given endpoint should have the same bus# for all functions.
SetCPTPointer Sets the internal configuration pointer.	cpt_ptr	I	Sets the internal configuration page table pointer. Typically called once in the Instantiation Model.
PopMemWriteNotification Pop the next memory write event in the queue (see MEM_WRITE_NOTIFICATION_ MODE parameter to enable.)	address (64)	O	Destination address.
	first_data_dword (32)	O	First data dword of the write.
	data_dword_len (16)	O	Data length of the write transaction.
	first_dw_be (4)	O	First data word byte enable
	last_dw_be (4)	O	Last data word byte enable
	flags (32)	O	Indicate details of this memory write. Currently defined flags are listed in the file *pciesvc_application_parms.v*
	valid (1)	O	Set if this event has valid data in it.

3.3.1.2 Completion Target Events

Completion target events are listed in Table 3-8.

Table 3-8 Completion target events

Event Name	Description
event_mem_write	This event will trigger after a memory write occurs in the completer. See related MEM_WRITE_NOTIFICATION_MODE parameter to control this event.

3.3.1.3 Completion Target I/O Space

The Completion Target also can access an I/O memory space allowing reads and write to I/O addresses. Like the memory, the VIP I/O memory is a *sparse* model, allowing a number of disparate addresses to be accessed with a small amount of actual memory. Tasks that provide I/O memory are listed in Table 3-9.

Table 3-9 Completion target I/O memory space access tasks

Task Name	Arguments (length)	I/O	Values
Write	address (32)	I	Target PCIE address. Must be dword aligned (bits [1:0] == 0).
	byte_enables (4)	I	Data word byte enables. The bits are in TLP bit order (bit0 corresponds to byte 3 of the data).
Write the provided data to the target PCIE I/O address, applying the byte enables.	data (32)	I	Data for the write transaction. The data is in TLP byte-order (byte0 on the “left” and byte3 on the “right”).
	status (32)	O	Returned status for the write.
Read	address (32)	I	Target PCIE address. Must be dword aligned (bits [1:0] == 0).
	byte_enables (4)	I	Data word byte enables. The bits are in TLP bit order (bit0 corresponds to byte 3 of the data).
Read the from the target PCIE I/O address into data, applying the byte enables.	data (32)	O	Data for the read transaction. The data is in TLP byte-order (byte0 on the “left” and byte3 on the “right”).
	status (32)	O	Returned status for the read.

3.3.1.4 Completion Target Configuration Space

The Completion Target has access to its own Configuration space allowing reads and writes to Configuration registers (including Capabilities). In contrast with the VIP Memory and I/O targets, the Configuration space is located in a fixed 4K sized *configuration block* (as defined in the PCIE spec.) This block includes not only the standard PCI configuration registers, but the extended space (including extended capabilities) defined by PCIE.

Each device will allocate a *Configuration Pointer Table* which contains pointers to all of the Configuration Blocks allocated (one for each function in the device).

Tasks that provide access to the completion target configuration space access are listed in Table 3-10.

Table 3-10 Completion target configuration space access tasks

Task Name	Arguments (length)	I/O	Values
BuildCfgPtrTable	max_func_num (32)	I	Maximum number of functions supported by this device.
	cfg_type (1)	I	Type0 or Type1 configuration block.*
Create a Configuration Pointer Table used to keep track of all the functions’ configuration information.	cpt_ptr (32)	O	Returned pointer to the Configuration Ptr Table

Table 3-10 Completion target configuration space access tasks (Continued)

Task Name	Arguments (length)	I/O	Values
AllocCfgBlock Allocate and create a Configuration register and capabilities block for a given function.	cpt_ptr (32)	I	The CPT pointer (returned from above call to *BuildCfgPtrTable*).
	func_num (32)	I	The function number for the created configuration block.
	cfg_type (1)	I	The configuration type (Type0 or Type1) that this created CfgBlock will be.*
	func_type (4)	I	The particular type of function this will be (PF, BF, VF, etc.)
	sr_iov_pf (8)	I	The PF that is the 'parent' Physical function of this VF.
	mr_iov_bf (8)	I	The BF that is the 'parent' Base Function of this function.
	status (32)	O	Returned status for the allocate.*
FillCfgBlock Mark the Cfg block as 'valid' and fill values in for a Capabilities Ptr as well as default values on the following Cfg registers: Command Reg Status Reg PCIe Capabilities Reg Device Capabilities Reg Device Control Capabilites Reg Note that this should typically only be called once (after calling AllocCfgBlock). Further configuration override can be done either via the below WriteCfgReg, or a PCIe CfgWr transaction to the bus/function.	cpt_ptr (32)	I	The CPT pointer for the device you want to fill.
	func_num (8)	I	The function number of the Configuration Block you want to fill.
WriteCfgReg Write the data to the target PCIE Configuration Space register number on the device specified by the cpt_ptr.	cpt_ptr (32)	I	The CPT pointer for the device you want to access.
	func_num (8)	I	The function number of the Configuration Block you want to access.
	cfg_reg (32)	I	The configuration-register number you are attempting to write to.*
	data (32)	I	Data for the configuration write transaction.
	status (32)	O	Returned status for the write.*

Table 3-10 Completion target configuration space access tasks (Continued)

Task Name	Arguments (length)	I/O	Values
ReadCfgReg Read from the target PCIE Configuration Space register number into data.	cpt_ptr (32)	I	The CPT pointer for the device you want to access.
	func_num (8)	I	The function number of the Configuration Block you want to access.
	cfg_reg (32)	I	The configuration register number you are attempting to read from.*
	data (32)	I	Data for the read transaction.
	status (32)	O	Returned status for the read.*
WriteCfgCapReg Write the data to the target PCIE Configuration Space register of the requested capability ID on the device specified by the cpt_ptr.	cpt_ptr (32)	I	The CPT pointer for the device you want to access.
	func_num (8)	I	The function number of the Configuration Block you want to access.
	cap_id (8)	I	The requested configuration-capability ID.*
	cfg_reg (32)	I	The configuration-register of the above capability that you are writing.*
	data (32)	I	Data for the configuration write transaction.
	status (32)	O	Returned status for the write.*
ReadCfgCapReg Read the data from the target PCIE Configuration Space register of the requested capability ID on the device specified by the cpt_ptr.	cpt_ptr (32)	I	The CPT pointer for the device you want to access.
	func_num (8)	I	The function number of the Configuration Block you want to access.
	cap_id (8)	I	The requested configuration-capability ID.*
	cfg_reg (32)	I	The configuration-register of the above capability that you are writing.*
	data (32)	O	Data for the configuration read transaction.
	status (32)	O	Returned status for the write.*
* - This field uses a parameter defined in Verilog/Application_PCIE/pciesvc_cfg_parms.v			

3.4 Target Memory

All PCIE devices in the system may have memory that is accessible by writes and/or reads to/from particular memory addresses. The VIP memory is a *sparse* model, allowing a wide variety of addresses (32 and 64-bit) to be accessed by a requester. The memory is divided into pages to increase performance, match up to PCIE packets and take advantage of locality-of-reference.

There are three page-sizes available for allocating pages such that the memory buffers are neither so large that memory is wasted nor are they inefficiently small.

The basic tasks are simply Write and Read, each providing dword-sized accesses via a supplied 32 or 64-bit PCIE address.

The three page sizes are simply: Large, Small and Dword (although the latter, being only 4-bytes long, hardly qualifies as a 'page'!) Since all accesses to the memory target are Dword at a time, it's difficult to predict the transaction's total data length; therefore a "hint" style reservation mechanism is used to allocate pages. When a PCIE memory write is intended, the client can call the task *PreWriteHint()* to request reservation of adequately sized pages. When the following Write() is called, the reserved pages are then used to efficiently store the data.

Note that it is up to the particular application using the target memory to choose optimal small and large page sizes based on their use-model. These applications are generally tuned to cooperate optimally with the DUT; similarly the memory target should be tuned to work with these applications. The task *DisplayStats()* can be useful in showing the allocated memory sizes during a simulation to help out in choosing optimal page sizes.

This memory is also used as the storage mechanism for the Global Shadow Memory (see "The Global PCIE Address Space Shadow" for details.)

3.4.1 Memory Target Parameters

Memory target parameters are listed in Table 3-11.

Table 3-11 Memory target parameters

Parameter Name	Settable VAR?*	Type	Range	Default	Description
PAGE_SIZE_IN_DWORDS					
	No	Integer	8-4096	64	Large page size, in units of dwords. Any transfer of this size (or larger) will allocate pages of this size.
SMALL_PAGE_SIZE_IN_DWORDS					
	No	Integer	8-256	8	Small page size, in units of dwords. Any transfer of this size (or larger, but less than the above PAGE_SIZE_IN_DWORDS) will allocate pages of this size. Any requests smaller than this will use individual Dwords.
NUM_64_BIT_PAGES					
	No	Integer	1024-16k	4096	Number of 64-bit pages initialized at startup. If the application attempts to allocate more pages than initialized, the allocation will fail, and the user should up the number of pages. Each large or small page or fword uses a single page entry.
NUM_32_BIT_PAGES					
	No	Integer	1024-16k	4096	Number of 32-bit pages initialized at startup. If the application attempts to allocate more pages than initialized, the allocation will fail, and the user should up the number of pages. Each large or small page or dword uses a single page entry.
NUM_ATTR_TBL_ENTRIES					

Table 3-11 Memory target parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
	No	Integer	1-1024	64	The number of attribute table entries. This defines how many memory ranges are available (see the Add/ RemoveMemRange task calls below).
DISPLAY_NAME					
	No	String		"memory_t arget0."	Default prefix in $msglog calls.
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with "_VAR". This is useful if using higher level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.					

3.4.1.1 Memory Target User Tasks

Memory target user tasks are listed in Table 3-12.

Table 3-12 Memory target user tasks

Task Name	Arguments (length)	I/O	Values
Write Write the provided data to the target PCIE memory address, applying the byte enables.	address(64)	I	Target PCIE address. Must be dword aligned (bits [1:0] == 0). If address[63:32] == 0, then it is considered a 32-bit address.
	byte_enables (4)	I	Data word byte enables. The bits are in TLP bit order (bit0 corresponds to byte 3 of the data).
	data (32)	I	Data for the write transaction. The data is in TLP byte-order (byte0 on the "left" and byte3 on the "right").
	status(32)	O	Returned status for the write.
Read Read the from the target PCIE memory address into data, applying the byte enables.	address(64)	I	Target PCIE address. Must be dword aligned (bits [1:0] == 0). If address[63:32] == 0, then it is considered a 32-bit address.
	byte_enables (4)	I	Data word byte enables. The bits are in TLP bit order (bit0 corresponds to byte 3 of the data).
	data (32)	O	Data for the read transaction. The data is in TLP byte-order (byte0 on the "left" and byte3 on the "right").
	status(32)	O	Returned status for the read. See Application_PCIE/ pciesvc_application_parms.v

Table 3-12 Memory target user tasks (Continued)

Task Name	Arguments (length)	I/O	Values
PreWriteHint Provide a hint to the memory target as to the length of the next write transaction.	address(64)	I	Base PCIE address for the upcoming write requests. Must be dword aligned (bits [1:0] == 0). This address should be the same as the first *Write* call that will be made following the call to PreWriteHint. If address[63:32] == 0, then it is considered a 32-bit address.
	dword_length (32)	I	Total length (in dwords) of the upcoming data transferred in this memory write transaction.
AddMemRange Define a special region of memory. Note that is only required if non-default attributes (e.g. IGNORED) are required on the memory range.	min_range (64)	I	Minimum memory address for the region.
	max_range (64)	I	Maximum memory address for the region.
	attributes (32)	I	The attribute(s) for the specified region.
	error (1)	O	Returned status for the AddMemRange. 0 indicates success.
RemoveMemRange Remove a special region of memory that was added previously. Note that the min/max_range must exactly match that which was specified in AddMemRange.	min_range(64)	I	Minimum memory address for the region.
	max_range (64)	I	Maximum memory address for the region.
	attributes (32)	I	The attribute(s) for the specified region to be removed.
	error (1)	O	Returned error status for the RemoveMemRange call.
DisplayStats Displays all stats in the Memory Target.	none		Displays all stats in the memory target.
ClearStats Clears stats counters in the Memory Target.	none		Clears all stats in the memory target.

4 Transaction Layer

The Transaction Layer encapsulates transactions generated by an application into TLPs. It also performs traffic class (TC) to virtual channel (VC) mapping, utilizes a credit-based flow control with the remote link and checks and enforces TLP ordering rules. VC0 is automatically initialized with default credits. In order to initialize VC1-7, SetInitTxCredits must be called followed by SetVcEnable.

4.1 Transaction Layer Module IOs

Transaction Layer module I/O signals are listed in Table 4-1.

Table 4-1 Transaction Layer module I/Os

Name	**I/O**	**Description**
reset	I(1)	Active high reset. Must only be asserted for 100ns ONCE per simulation at the beginning.
dl_status	I (32)	Bits (use predefined parms for access): [0] = link_up. [8] = VC0 initialized [9] = VC1 initialized [10] = VC2 initialized [11] = VC3 initialized [12] = VC4 initialized [13] = VC5 initialized [14] = VC6 initialized [15] = VC7 initialized

4.2 Transaction Layer Configuration Tasks

Transaction Layer configuration tasks are listed in Table 4-2.

Table 4-2 Transaction Layer configuration tasks

Task Name	Arguments	I/O	Values
SetVcEnable	vh (32)	I	Virtual Hierarchy number
	vc (32)	I	Virtual Channel number
	enable	I	Enable or disable vc
SetTrafficClassMap	tc(32)	I	Traffic class to be mapped to a virtual channel
	vc(32)	I	Virtual channel that traffic class is to be mapped to
	enable(1)	I	Enable bit for mapping.
SetInitTxCredits These credits will be sent to the link partner during credit initialization. This task must be called before SetVcEnable for VCs 1 – 7. Setting header/data credits to 0 indicates infinite credits.	vh(32)	I	Virtual Hierarchy
	vc(32)	I	Virtual channel associated with these credits
	credit_type(32)	I	Definitions (user pre-defined parms for access) 2'b00 = Posted 2'b01 = Non-Posted 2'b10 = Completion 2'b11 = Reserved
	hdr_credits(32)	I	Number of header credits for this credit_type
	data_credits(32)	I	Number of data credits for this credit_type

4.3 Transaction Layer User Tasks

Transaction Layer user tasks are listed in Table 4-3.

Table 4-3 Transaction Layer user tasks

Task Name	Arguments	I/O	Values
IsTransactionLayerIdle This task should be called at the end of each test.	result (1)	O	Is idle is all queues are empty
AddMemAddrApplIdMapEntry Used to map memory addresses to an application.	memory_addr(64)	I	Base Address of memory range
	memory_window(64)	I	Window of addresses which will cause match of entry.
	appl_id(32)	I	Application ID to map TLP to.
	error(1)	O	Indication that addition of new entry failed.
AddIOAddrApplIdMapEntry Used to map I/O addresses to an application.	memory_addr(64)	I	Base Address of memory range
	memory_window(64)	I	Window of addresses which will cause match of entry.
	appl_id(32)	I	Application ID to map TLP to.
	error(1)	O	Indication that addition of new entry failed.

Table 4-3 Transaction Layer user tasks (Continued)

Task Name	Arguments	I/O	Values
AddCfgBDFApplIdMapEntry Used to map Config Request {Bus, Device, Function} to applications. Used when VIP is Upstream port of a link. This mapping is enabled via the ENABLE_ROUTE_CFG_ TYPE[0|1]_TO_ FUNCTION parameter.	config_type(1)	I	If true, Configuration is a Type 1 request. If false, Configuration is a Type 0 request.
	bdf(16)	I	{bus, device, function} of request.
	appl_id(32)	I	Application ID to map TLP to.
	error(1)	O	Indication that addition of new entry failed.
AddATAddrApplIdMapEntry Used to map memory addresses which need address translation to an application.	memory_addr(64)	I	Base Address of memory range
	memory_window(64)	I	Window of addresses which will cause match of entry.
	appl_id(32)	I	Application ID to map TLP to.
	error(1)	O	Indication that addition of new entry failed.
AddRIdMsgCodeApplIdMapEntry Used to map {Requester Ids, msgcode} of MSG TLPs to applications.	requester_id(16)	I	Requester Id to map
	msgcode(8)	I	Msgcode of TLP. Same value as msgcode field in TLP header. See Include/pciesvc_parms.v file for defines.
	appl_id(32)	I	Application ID to map TLP to.
	error(1)	O	Indication that addition of new entry failed.
AddRequesterIdApplIdMapEntry Used to map Requester Ids to applications. This table is automatically populated when TLPs are sent by the Transaction Layer. This mapping is always enabled.	requester_id(16)	I	Requester Id to map
	appl_id(32)	I	Application ID to map TLP to.
	error(1)	O	Indication that addition of new entry failed.
DisplayMemAddrApplidMap Displays all memory address to application id map entries.	none		Displays all entries in Memory Address mapping table.
DisplayIOAddrApplidMap Displays all I/O address to application id map entries.	none		Displays all entries in I/O Address mapping table.
DisplayCfgBDFApplidMap Displays all Configuration Type 0 and Type 1 {Bus, Device, Function} to application id map entries. Use when VIP is Upstream port of a link.	none		Displays all entries in CfgBDF mapping table.

Table 4-3 Transaction Layer user tasks (Continued)

Task Name	Arguments	I/O	Values
DisplayATAddrApplidMap Displays all memory address to application id map entries which require address translation.	none		Displays all entries in AT Address mapping table.
DisplayRidMsgCodeApplidMap Displays all {Requester Id, MsgCode} to application id map entries.	none		Displays all entries in RidMsgCode mapping table.
DisplayRidApplidMap Displays all Requester Id to application id map entries. Use when VIP is Upstream port of a link.	none		Displays all entries in Requestor id mapping table.
DisplayStats Displays all stats in Transaction Layer.	none		Displays all stats in Transaction Layer.
ClearStats Clears all stats in Transaction Layer.	none		Clear all stats in Transaction Layer.
IsVcInitFinished Used to determine if Data Link Layer has finished the Init Flow Control handshake for a particular VC.	vh(32)	I	Virtual Hierarchy.
	vc(32)	I	Virtual channel to check.
QueryRxCreditsAvailable Returns credits this VIP has received and are available for sending TLPs.	vh(32)	I	Virtual Hierarchy
	vc(32)	I	Virtual channel to check.
	fc_type	I	Definitions (user pre-defined parms for access) 2'b00 = Posted 2'b01 = Non-Posted 2'b10 = Completion 2'b11 = Reserved
	avail_hdr_credits	O	Actual number header credits available for transmitting TLPs. Limit – consumed header credit values.
	infinite_hdr_credits	O	If set, TLPs are not gated due to header credits.
	avail_data_credits	O	Actual number data credits available for transmitting TLPs. Limit – consumed data credit values.
	infinite_data_credits	O	If set, TLPs are not gated due to data credits.

Table 4-3 Transaction Layer user tasks (Continued)

Task Name	Arguments	I/O	Values
QueryTxCreditsAvailable Returns number of credits transmitted to the link partner for sending TLPs.	vh(32)	I	Virtual Hierarchy
	vc(32)	I	Virtual channel to check.
	fc_type	I	Definitions (user pre-defined parms for access) 2'b00 = Posted 2'b01 = Non-Posted 2'b10 = Completion 2'b11 = Reserved
	avail_hdr_credits	O	Actual number header credits transmitted to link partner. Link partner can use these credits to send TLPs. Allocated – Received header credit values.
	infinite_hdr_credits	O	If set, initial header credits sent to link partner were infinite.
	avail_data_credits	O	Actual number data credits transmitted to link partner. Link partner can use these credits to send TLPs. Allocated – Received data credit values.
	infinite_data_credits	O	If set, initial data credits sent to link partner were infinite.

Table 4-3 Transaction Layer user tasks (Continued)

Task Name	Arguments	I/O	Values
QueryCreditCounts Returns actual counter values as selected by credit_sel. These are the values in the counters themselves and not the actual number of credits available.	vh(32)	I	Virtual Hierarchy
	vc(32)	I	Virtual channel to check.
	credit_sel	I	Definitions (user pre-defined parms for access) 0 = init allocated 1 = allocated 2 = limit 3 = received 4 = consumed 5 = init limit
	credit_type	I	Definitions (user pre-defined parms for access) 2'b00 = Posted 2'b01 = Non-Posted 2'b10 = Completion 2'b11 = Reserved
	hdr_credits	O	Value stored in header counter indicated by credit_sel. Not actual number of credits.
	data_credits	O	Value stored in data counter indicated by credit_sel. Not actual number of credits.
CheckFinalCredits Compares initial allocated credits to final allocated – received credit values. Compares initial Limit credits to final limit – consumed credit values. Warnings are issued if any credits are lost.	n/a	n/a	This task should be called at the end of every test. Any lost credits will be flagged.

4.4 Transaction Layer Parameters

Transaction Layer parameters are listed in Table 4-4.

Table 4-4 Transaction Layer parameters

Parameter Name	Settable VAR?*	Type	Range	Default Value	Description
PCIE_SPEC_VER					
	Yes	Real	1.1, 2,0, 2.1, 3.0	PCIE_SPEC_VER_2_1	See Include/pciesvc_parms.v: PCIE_SPEC_VER_* Note: Please set this in the model.
ENABLE_ROUTE_MEM_TO_ FUNCTION					
	Yes	Integer	0-1	0	Enable routing memory requests to function via memory request mapping table. Use task AddMemAddrApplIdMapEntry to program table.

Table 4-4 Transaction Layer parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default Value	Description
DEFAULT_ROUTE_MEM_APPL_ ID					
	Yes	Integer	0-2^16	0	Default Application ID to use if above enable is not asserted or mapping fails.
ENABLE_ROUTE_IO_TO_ FUNCTION					
	Yes	Integer	0-1	0	Enable routing I/O requests to function via I/O request mapping table. Use task AddIOAddrApplIdMapEntry to program table.
DEFAULT_ROUTE_IO_APPL_ID					
	Yes	Integer	0 - large value	0	Default Application ID to use if above enable is not asserted or mapping fails.
ENABLE_ROUTE_CFG_TYPE0_ TO_FUNCTION					
	Yes	Integer	0-1	0	Enable routing Type 0 Configuration requests to function via configuration request mapping table. Task AddCfgBDFApplIdMapEntry used to program table.
DEFAULT_ROUTE_CFG_TYPE0_APPL_ID					
	Yes	Integer	0 - large value	0	Default Application ID to use if corresponding Type 0 enable listed above is not asserted or mapping fails.
ENABLE_ROUTE_CFG_TYPE1_ TO_FUNCTION					
	Yes	Integer	0-1	0	Enable routing Type 1 Configuration requests to function via configuration request mapping table. Task AddCfgBDFApplIdMapEntry used to program table.
DEFAULT_ROUTE_CFG_TYPE1_APPL_ID					
	Yes	Integer	0 - large value	0	Default Application ID to use if corresponding Type 1 enable listed above is not asserted or mapping fails.
ENABLE_ROUTE_MSG_TO_ FUNCTION					
	Yes	Integer	0-1	0	Enable routing message requests to function via message request mapping table. If not enabled, the default Appl Id used for routing is as defined by DEFAULT_ROUTE_MSG_APPL_ ID
DEFAULT_ROUTE_MSG_APPL_ ID					
	Yes	Integer	0 - large value	0	Default Application ID to use if above enable is not asserted or mapping fails.

Table 4-4 Transaction Layer parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default Value	Description
ENABLE_ROUTE_AT_TO_ FUNCTION					
	Yes	Integer	0-1	0	Enable routing AT requests to function via AT request mapping table. Use task AddATApplIdMapEntry to program table.
DEFAULT_ROUTE_AT_APPL_ID					
	No	Integer	0 - large value	0	Default Application ID to route Address Translation requests to.
NUM_APPL_ID					
	No	Integer	8-128	8	Max number of unique Application IDs. IDs assigned to applications must be less than this value.
RID_APPLID_TABLE_SIZE					
	No	Integer	4-4096	64	Number of unique RID to Appl_id map entries.
RID_MSGCODE_APPLID_TABLE_SIZE					
	No	Integer	4-4096	64	Number of unique {RID,msgcode} to Appl_id map entries.
MEM_ADDR_ADDPLID_TABLE_SIZE					
	No	Integer	4-4096	64	Number of unique Mem Address to Appl_id map entries.
IO_ADDR_ADDPLID_TABLE_SIZE					
	No	Integer	4-4096	64	Number of unique I/O Address to Appl_id map entries.
AT_ADDR_ADDPLID_TABLE_SIZE					
	No	Integer	4-4096	64	Number of unique AT Address to Appl_id map entries.
AUTO_ENABLE_VC0_AT_STARTUP					
	Yes	Integer	0-1	1	Enable of VC0 at startup. VC0 automatically initializes unless this parm/_VAR is set to 0.

Table 4-4 Transaction Layer parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default Value	Description
NUM_VC_[P/NPCPL]_NIT_[HDRDATA]_CREDITS					
	Yes	Integer	Hdr = 0-255 Date = 0-4096	P_HDR =102 P_DATA =1024 NP_HDR =101 NP_DATA =1025 CPL_HDR =100 CPL_DATA =1026	Initial credits sent to link partner if SetAllocatedCredits is not called. For VC0 only.
MIN_VC[0-7]_[P/NP/CPL]_ UPDATEFC_DELAY					
	Yes	Integer		1 ns	Min delay from rx of TLP to tx of corresponding credit. This can be used to model delay associated with freeing rx TLP buffer space.
MAX_VC[0-7]_[P/NP/CPL]_ UPDATEFC_DELAY					
	Yes	Integer		100 ns	Max delay from rx of TLP to tx of corresponding credit. This can be used to model delay associated with freeing rx TLP buffer space.
CREDIT_STARVATION_TIMEOUT_NS					
	Yes	Integer		10,000 ns	If a VC is credit starved, i.e. TLP transmission is gated, for too long, a warning is issue. Credits should be returned in a timely fashion. Set to 0 to disable check.
MAX_NUM_END_TO_END_PREFIXES					
	Yes	Integer		4	Max number of prefixes allowed. Version 3 only.
REMOTE_MAX_PAYLOAD_SIZE					
	Yes	Integer		128 bytes	Remote Max Payload Size. Used to check incoming TLP size.
REMOTE_EXTENDED_TAG_FIELD_ENABLED					
	Yes	Integer	0-1	0	Remote extended Tag enabled support as per Config 7.8.4. Used to check incoming TLPs.
REMOTE_MAX_READ_REQUEST_SIZE					
	Yes	Integer		512 bytes	Remote Max Read Request size. Used to check incoming Read Request TLP size.
ENABLE_ARI					
	Yes	Integer	0-1	0	Enable ARI in Transaction Layer. Version 3 only.

Table 4-4 Transaction Layer parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default Value	Description
IS_TX_DOWNSTREAM					
	No	Integer	0-1	0	Stack direction. Used for TLP header checking/ routing.
IS_SWITCH					
	No	Integer	0-1	0	Is this stack part of a switch? Used for TLP header checking/routing.
DISPLAY_VC_QUEUES_PERIOD_NS					
	Yes	Integer	10_000 - large value	50000	Display contents of enabled VC queues. Setting to 0 disables the display.
ENABLE_SHADOW_CFG_LOOKUP					
	Yes	Integer		1	When this is 1, the Transaction Layer will examine the shadow configuration database of the source of received TLPs to allow TLP checks against that configuration. Make sure this is not enabled if the shadow memory is not instantiated.
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with "_VAR". This is useful if using higher level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.					

4.5 Transaction Layer Events

Transaction Layer events are listed in Table 4-5.

Table 4-5 Transaction Layer events

Event Name	Description
event_sent_credit	Credits were sent.
event_received_credit	Credits were received.
event_sent_tlp	TLP was sent.
event_received_tlp	TLP was received.

4.6 Transaction Layer ASCII Signals

Transaction Layer ASCII signals are listed in Table 4-6.

Table 4-6 Transaction Layer ASCII Signals

Event Name	Description
ascii_tx_tlp_type	Type of TLP to be sent as defined by {fmt, type} fields.
ascii_tx_tlp_fc_type	Flow control credits associated with this TLP. Values are P, NP, CPL.
ascii_tx_tlp_vc	VC TLP sent on.
ascii_tx_tlp_xid	Sent TLP transaction Id
ascii_rx_tlp_type	Type of received TLP as defined by {fmt, type} fields.
ascii_rx_tlp_fc_type	Flow control credits associated with this TLP. Values are P, NP, CPL.
ascii_rx_tlp_vc	VC TLP received on.
ascii_rx_tlp_xid	Received TLP transaction Id

4.7 Transaction Layer Statistics

Transaction Layer statistics are listed in Table 4-7.

Table 4-7 Transaction Layer statistics

Stat Name	Width	Description
stat_num_tlps_sent	32	Number of TLPs sent by the link
stat_num_tlps_received	32	Number of TLPs received by the link
stat_num_good_tlps_sent	32	Number of TLPs sent without error
stat_num_good_tlps_received	32	Number of TLPs received without error
stat_num_bad_tlp_sent	32	Number of TLPs sent with error
stat_num_bad_tlp_received	32	Number of TLPs received with error
stat_num_tc[0-7]_tlps_sent	32	Number of tc[0-7] TLPs sent
stat_num_tc[0-7]_tlps_received	32	Number of tc[0-7] TLPs received
stat_num_vc[0-7]_tlps_sent	32	Number of vc[0-7] TLPs sent
stat_num_vc[0-7]_tlps_received	32	Number of vc[0-7] TLPs received
stat_num_tlp_cfg_rd0_received	32	Number of Cfg rd0 TLPs received
stat_num_tlp_cfg_wr0_received	32	Number of Cfg wr0 TLPs received
stat_num_tlp_cfg_rd1_received	32	Number of Cfg rd1 TLPs received
stat_num_tlp_cfg_wr1_received	32	Number of Cfg wr1 TLPs received
stat_num_tlp_io_rd_received	32	Number of I/O rd TLPs received
stat_num_tlp_io_wr_received	32	Number of I/O wr TLPs received
stat_num_tlp_tcfg_rd_received	32	Number of TCfg rd0 TLPs received
stat_num_tlp_tcfg_wr_received	32	Number of TCfg wr0 TLPs received
stat_num_tlp_msg_received	32	Number of msg TLPs received
stat_num_tlp_msgd_received	32	Number of MsgD TLPs received

Table 4-7 Transaction Layer statistics (Continued)

Stat Name	Width	Description
stat_num_tlp_msg_assert_int[a-d]_received	32	Number of msg assert Int[a-d] TLPs received
stat_num_tlp_msg_deassert_int[a-d]_received	32	Number of msg deassert Int[a-d] TLPs received
stat_num_tlp_pm_active_state_nak_received	32	Number of msg pm active state NAK received
stat_num_tlp_pm_pme_received	32	Number of msg pme received
stat_num_tlp_pm_turn_off_received	32	Number of msg pm turn off received
stat_num_tlp_pm_to_ack_received	32	Number of msg pm to ACK received
stat_num_tlp_err_cor_received	32	Number of msg correctable error received
stat_num_tlp_err_nonfatal_received	32	Number of msg nonfatal error received
stat_num_tlp_err_fatal_received	32	Number of msg fatal error received
stat_num_tlp_unlock_received	32	Number of msg unlock received
stat_num_tlp_set_pwr_limit_received	32	Number of msg set pwr limit received
stat_num_tlp_vendor_type0_received	32	Number of msg vendor type0 received
stat_num_tlp_vendor_type1_received	32	Number of msg vendor type1 received
stat_num_tlp_ltr_received	32	Number of msg ltr received
stat_num_tlp_obff_received	32	Number of msg obff received
stat_num_tlp_mem_wr32_received	32	Number of TLP Mem wr 32 received
stat_num_tlp_mem_wr64_received	32	Number of TLP Mem wr 64 received
stat_num_tlp_mem_rd32_received	32	Number of TLP Mem rd 32 received
stat_num_tlp_mem_rd64_received	32	Number of TLP Mem rd 64 received
stat_num_tlp_mem_rd_lock32_received	32	Number of TLP Mem rd lock 32 received
stat_num_tlp_mem_rd_lock64_received	32	Number of TLP Mem rd lock 64 received
stat_num_tlp_fetch_add32_received	32	Number of TLP fetch add32 received
stat_num_tlp_fetch_add64_received	32	Number of TLP fetch add64 received
stat_num_tlp_swap32_received	32	Number of TLP swap32 received
stat_num_tlp_swap64_received	32	Number of TLP swap64 received
stat_num_tlp_cas32_received	32	Number of TLP cas32 received
stat_num_tlp_cas64_received	32	Number of TLP cas64 received
stat_num_tlp_cpl_received	32	Number of TLP cpl received
stat_num_tlp_cpld_received	32	Number of TLP cpld received
stat_num_tlp_cpl_lock_received	32	Number of TLP cpl lock received
stat_num_tlp_cpld_lock_received	32	Number of TLP cpld lock received
stat_num_tlp_cfg_rd0_sent	32	Number of cfg rd0 TLPs sent
stat_num_tlp_cfg_wr0_sent	32	Number of cfg wr0 TLPs sent
stat_num_tlp_cfg_rd1_sent	32	Number of cfg rd1 TLPs sent

Table 4-7 Transaction Layer statistics (Continued)

Stat Name	Width	Description
stat_num_tlp_cfg_wr1_sent	32	Number of cfg wr1 TLPs sent
stat_num_tlp_io_rd_sent	32	Number of I/O rd TLPs sent
stat_num_tlp_io_wr_sent	32	Number of I/O wr TLPs sent
stat_num_tlp_tcfg_rd_sent	32	Number of TCfg rd0 TLPs sent
stat_num_tlp_tcfg_wr_sent	32	Number of TCfg wr0 TLPs sent
stat_num_tlp_msg_sent	32	Number of msg TLPs sent
stat_num_tlp_msgd_sent	32	Number of MsgD TLPs sent
stat_num_tlp_msg_assert_int[a-d]_sent	32	Number of msg assert int[a-d] TLPs sent
stat_num_tlp_msg_deassert_int[a-d]_sent	32	Number of msg deassert int[a-d] TLPs sent
stat_num_tlp_pm_active_state_nak_sent	32	Number of msg pm active state NAK sent
stat_num_tlp_pm_pme_sent	32	Number of msg pme sent
stat_num_tlp_pm_turn_off_sent	32	Number of msg pm turn off sent
stat_num_tlp_pm_to_ack_sent	32	Number of msg pm to ACK sent
stat_num_tlp_err_cor_sent	32	Number of msg correctable error sent
stat_num_tlp_err_nonfatal_sent	32	Number of msg nonfatal error sent
stat_num_tlp_err_fatal_sent	32	Number of msg fatal error sent
stat_num_tlp_unlock_sent	32	Number of msg unlock sent
stat_num_tlp_set_pwr_limit_sent	32	Number of msg set pwr limit sent
stat_num_tlp_vendor_type0_sent	32	Number of msg vendor type0 sent
stat_num_tlp_vendor_type1_sent	32	Number of msg vendor type1 sent
stat_num_tlp_ltr_sent	32	Number of msg ltr sent
stat_num_tlp_obff_sent	32	Number of msg obff sent
stat_num_tlp_mem_wr32_sent	32	Number of TLP Mem wr 32 sent
stat_num_tlp_mem_wr64_sent	32	Number of TLP Mem wr 64 sent
stat_num_tlp_mem_rd32_sent	32	Number of TLP Mem rd 32 sent
stat_num_tlp_mem_rd64_sent	32	Number of TLP Mem rd 64 sent
stat_num_tlp_mem_rd_lock32_sent	32	Number of TLP Mem rd lock 32 sent
stat_num_tlp_mem_rd_lock64_sent	32	Number of TLP Mem rd lock 64 sent
stat_num_tlp_fetch_add32_sent	32	Number of TLP fetch add32 sent
stat_num_tlp_fetch_add64_sent	32	Number of TLP fetch add64 sent
stat_num_tlp_swap32_sent	32	Number of TLP swap32 sent
stat_num_tlp_swap64_sent	32	Number of TLP swap64 sent
stat_num_tlp_cas32_sent	32	Number of TLP cas32 sent
stat_num_tlp_cas64_sent	32	Number of TLP cas64 sent

Table 4-7 Transaction Layer statistics (Continued)

Stat Name	Width	Description
stat_num_tlp_cpl_sent	32	Number of TLP cpl sent
stat_num_tlp_cpld_sent	32	Number of TLP cpld sent
stat_num_tlp_cpl_lock_sent	32	Number of TLP cpl lock sent
stat_num_tlp_cpld_lock_sent	32	Number of TLP cpld lock sent
stat_num_dl_status_down_events	32	Number of dl status down events

4.8 Transaction Layer Internal Interface Tasks

The Transaction Layer interfaces with the application layer above and with the Data Link Layer below. Application layer to Transaction Layer interface tasks are listed in Table 4-8. Transaction Layer to Data Link Layer tasks are listed in Table 4-10.

Table 4-8 Application layer – Transaction Layer interface tasks

Task Name	Arguments	I/O	Values
SendPacket Task pair called with task of same name in application. Used to send TLPs.	appli_id(32)	I	Application ID TLP was sent from.
	tlp_vh(32)	I	Virtual hierarchy to send TLP on.
	tlp_tc(32)	I	Traffic class of the outgoing packet
	tlp_ptr(32)	I	Pointer to the TLP header
	tlp_len(32)	I	Length of the TLP in dwords
	tlp_ei_code(32)	I	Error injection code. Error injection code can be found in Verilog/Include/pciesvc_parms.v and start with the prefix TX_TLP_EI_*
	req(1)	I	Request to send TLP
	ack(1)	O	Acknowledgement that request was accepted.
ReceivePacket Task pair called with task of same name in application. Used to receive TLPs from VIP.	appl_id(32)	I	Application ID requesting TLP.
	tlp_vh(32)	O	Virtual hierarchy TLP was received on.
	tlp_tc(32)	O	Traffic class of the received TLP.
	tlp_ptr(32)	O	Pointer to data in the received TLP. NOTE: This pointer must be freed by the user to avoid memory leakage. See the FreeMemory task description in "Memory Model Utility"
	tlp_len(32)	O	Length of the TLP in dwords.
	tlp_ei_code(32)	O	Error injection code. No rx error injection codes are currently defined.
	tlp_status(32)	O	Error status of TLP. Used by Application to generate the appropriate Completion TLP.
	req(1)	O	Request to pass the received packet up.
	ack(1)	I	Acknowledgement that the received packet was accepted.

There are some error injection codes that are not inserted using automated error injection but can be injected using either the Synopsys user application or a user-created application. It is up to you to inject these errors when you create a TLP, but the appropriate error injection code must be used so that lower layers know the TLP has an error. The error injection codes are described in Table 4-9.

Table 4-9 Error injection codes

Error Inection Code	**Description**
TX_TLP_EI_INVALID_FMT_TYPE	The TLP has a reserved or undefined value on the fmt/type fields.
TX_TLP_EI_CORRUPT_LENGTH_FIELD	The length of the TLP does not match the length field.
TX_TLP_EI_CORRUPT_ECRC	The end-to-end cyclic redundancy check value is corrupted.
TX_TLP_EI_CORRUPT_FIRST_DW_BE	The first DW BE is set to a reserved/illegal value.
TX_TLP_EI_CORRUPT_LAST_DW_BE	The last DW BE is set to a reserved/illegal value.
TX_TLP_EI_INVALID_ATTR	The attribute field is set to a value with is invalid/illegal for the TLP type.
TX_TLP_EI_INVALID_AT	The AT bit is set to a value which is invalid/illegal for the TLP type.
TX_TLP_EI_INVALID_TH	The TH bit is set to a value which is invalid/illegal for the TLP type.
TX_TLP_EI_MALFORMED	The TLP is malformed in some other way not specified by the above error injection codes.

Transaction Layer <-> Data Link Layer interface tasks are described in Table 4-10.

Table 4-10 Transaction Layer – Data Link Layer interface tasks

Task Name	**Arguments**	**I/O**	**Values**
TransmitTLP Task pair called with task of same name in Data Link Layer. Used to send TLPs.	tlp_ptr(32)	O	Pointer of the TLP to be sent.
	tlp_len(32)	O	Length of the data buffer containing the TLP in dwords.
	tlp_ei_code(32)	O	Error injection code.
	req	O	Request from application layer to Transaction Layer
	ack	I	Acknowledgement that Transaction Layer has accepted the transmission request from the application.
ReceiveTLP Task pair called with task of same name in Data Link Layer. Used to receive TLPs.	tlp_vh	O	
	tlp_vc	O	
	tlp_ptr(32)	I	Pointer to the sent TLP.
	tlp_len(32)	I	Length of the sent TLP buffer in dwords.
	ei_code(32)	I	Error injection code. No EI codes defined for receive side.
	Req	I	Request to from Transaction Layer to application layer to pass TLP contents.
	Ack	O	Acknowledgement that TLP contents have been accepted by the application layer.

Table 4-10 Transaction Layer – Data Link Layer interface tasks (Continued)

Task Name	Arguments	I/O	Values
TransmitCredits Task pair called with task of same name in Data Link Layer. Used to send Credits. Instruct the link to send an UpdateFCP or InitFCP DLLP.	vh(32)	O	Virtual hierarchy to send DLLP to.
	vc(32)	O	Virtual channel to send DLLP to.
	credit_type(32)	O	update_type[1:0] == 2'b00 means P update update_type[1:0] == 2'b01 means Non-Posted update update_type[1:0] == 2'b10 means Completion type.
	hdr_credits(32)	O	Number of header credits sent
	data_credits(32)	O	Number of data credits sent
	credit_is_init	O	This is the first credits sent. Will be use in FC Initialization process.
	req(1)	O	Request to link
	ack(1)	I	Acknowledgement from link that request was accepted.
ReceiveCredits Task pair called with task of same name in Data Link Layer. Used to receive credits from Data Link Layer.	vh(32)	I	Virtual hierarchy DLLP sent on.
	vc(32)	I	Virtual channel DLLP sent on.
	credit_type(32)	I	update_type[1:0] == 2'b00 means P update update_type[1:0] == 2'b01 means Non-Posted update update_type[1:0] == 2'b10 means Completion type.
	hdr_credits(32)	I	Number of header credits sent
	data_credits(32)	I	Number of data credits sent
	credit_is_init	I	This is the first credits sent. These are initialization credits.
	req(1)	I	Request to link
	ack(1)	O	Acknowledgement from link that request was accepted.

5 Data Link Layer

The Data Link Layer accepts TLPs from the Transaction Layer and schedules them for transmission., as well as conveying transmitted TLPs to the Transaction Layer. It is also responsible for sourcing and sinking DLLPs. TLPs can be copied to a scoreboard via SentTLP and SentTLP tasks.

5.1 Date Link Layer Module I/Os

I/Os of the Data Link Layer module are listed in Table 5-1.

Table 5-1 Data Link Layer module I/Os

Name	I/O	Description
reset	I (1)	Active high reset. Must only be asserted for 100ns ONCE per simulation at the beginning.
link_width	I (32)	Link width from Phy
phy_status	I (32)	Bit definitions (use predefined bit parms for access) [0] = link_up. [1] = phy in recovery
freq_select	I (32)	Frequency select from Phy
dl_status	O (32)	Bit definitions (use predefined bit parms for access) [0] = link_up. [8] = VC0 initialized [9] = VC1 initialized [10] = VC2 initialized [11] = VC3 initialized [12] = VC4 initialized [13] = VC5 initialized [14] = VC6 initialized [15] = VC7 initialized
retrain_link	O (1)	Instruction from the Data Link Layer to the Physical Layer that it needs to retrain the phy. Each bit corresponds to a link.

5.2 Data Link Layer Configuration Tasks

Data Link Layer configuration tasks are listed in Table 5-2.

Table 5-2 Data Link Layer configuration tasks

Task Name	Arguments	I/O	Values
SetLinkEnable This task enables the Data Link Layer. The Data Link Layer will automatically initiate VC0 InitFC process when enabled.	enable(1)	I	Set to 1 to enable the link

5.3 Data Link Layer User Tasks

Data Link Layer user tasks are listed in Table 5-3.

Table 5-3 User Specific – Link Interface

Task Name	Arguments (length)	I/O	Values
IsDataLinkIdle This task should be called at the end of each test. Returns 1 if all queues are empty and no EIs are in progress.	result5	O	Returns 1 if no TLP or DLLP requests pending.
DisplayStats Displays all stats in Data Link Layer.	none		Displays all stats in Data Link Layer
ClearStats Clears all stats in Data Link Layer.	none		Clear all stats in Data Link Layer
TransmitUserDLLP Allows user to send DLLPs.	user_data (32)	I	Unframed DLLP
	user_ei_code (32)	I	Error injection code
	valid	I	Request from Vendor logic to Data Link Layer.
	block	I	1 = Task will not return until DLLP has been transmitted. 0 = DLLP will be posted to a queue which will schedule DLLP for transmission. Task will return immediately.
ReceiveVendorDLLP Allows user to pick up incoming vender specific DLLPs.	vendor_data (24)	O	
	vendor_vc (32)	O	Virtual channel number.
	vendor_status (32)	I	Receive status
	req	O	Request from Data Link Layer to Vendor logic.
	ack	I	Acknowledgement that vendor logic has accepted the receive request from the Data Link.

Table 5-3 User Specific – Link Interface (Continued)

Task Name	Arguments (length)	I/O	Values
SentTLP Scoreboard interface. If enabled via SENT_TLP_ INTERFACE_MODE, all TLPs sent by Data Link Layer will be queued and accessed via this task. event_sent_tlp can be used to wait for valid TLPs to be queued. See "Data Link Layer Events". *NOTE: For scoreboards attached to this interface, any TLP with ei_code == TX_TLP_EI_NONE will be received by the DUT's application layer.*	ptr (32)	O	Pointer to TLP stored in memory. NOTE: This pointer must be freed by the user to avoid memory leakage. See the FreeMemory task description in "Memory Model Utility".
	len (32)	O	Length of TLP stored in memory in dwords.
	seq_num (32)	O	Bits [11:0] – Sequence number of TLP
	lcrc (32)	O	LCRC of sent TLP
	ei_cod (32)	O	Error Injection code of TLP sent. See "Data Link LayerTransmit TLP Error Injection" for a list of the codes.
	transmit_status (32)	O	Status of the sent TLP. SENT_TLP_STATUS_GOOD_BIT is asserted for each TLP DUT is expected to forward to the TL. Any other bit asserted Indicates VIP Data Link Layer injected error. The TLP with errors must be dropped by the DUT. Dropped TLPs may be retried by VIP. See Include/pciesvc_parms.v for bit definitions SENT_TLP_STATUS_*_BIT.
	valid (1)	O	Pointer is valid
ReceivedTLP Scoreboard interface. If enabled via RECEIVED_TLP_ INTERFACE_MODE, all TLPs received by the Data Link Layer will be queued and accessed via this task. event_received_tlp can be used to wait for valid TLPs to be queued. See "Data Link Layer Events".	ptr (32)	O	Pointer to TLP stored in memory. NOTE: This pointer must be freed by the user to avoid memory leakage. See the FreeMemory task description in "Memory Model Utility".
	len (32)	O	Length of TLP stored in memory in dwords.
	seq_num (32)	O	Bits [11:0] – Sequence number of TLP
	lcrc (32)	O	LCRC of received TLP
	ei_code (32)	O	Error Injection code of TLP received. "Data Link LayerTransmit TLP Error Injection" for a list of the codes.
	received_status (32)	O	Status of the received TLP. RECEIVED_TLP_STATUS_GOOD_BIT is asserted for each TLP VIP forwards to the TL. Any other bit asserted Indicates TLP was received with error. TLPs with errors will be dropped, but possible retried by DUT. See Include/pciesvc_parms.v for bit definitions RECEIVED_TLP_STATUS_*_BIT.
	valid (1)	O	Pointer is valid.

Table 5-3 User Specific – Link Interface (Continued)

Task Name	Arguments (length)	I/O	Values
SentDLLP Scoreboard interface. If enabled via SENT_DLLP_ INTERFACE_MODE, all DLLPs sent by the Data Link LayerData Link Layerwill be queued and accessed via this task. event_sent_dllp can be used to wait for valid DLLPs to be queued. See "Data Link Layer Events".	ptr (32)	O	Pointer to DLLP stored in memory. NOTE: This pointer must be freed by the user to avoid memory leakage. See the FreeMemory task description in "Memory Model Utility".
	len (32)	O	Length of DLLP stored in memory in dwords.
	ei_code (32)	O	Error Injection code of DLLP sent. See "Data Link Layer Transmit DLLP Error Injection" for a list of the codes.
	transmit_status (32)	O	Status of the sent DLLP. See Include/ pciesvc_parms.v for bit definitions SENT_DLLP_STATUS_*_BIT.
	valid (1)	O	Pointer is valid.
ReceivedDLLP Scoreboard interface. If enabled via RECEIVED_DLLP_ INTERFACE_MODE, all DLLPs received by the Data Link LayerData Link Layerwill be queued and accessed via this task. event_received_dllp can be used to wait for valid DLLPs to be queued. See "Data Link Layer Events".	ptr (32)	O	Pointer to DLLP stored in memory. NOTE: This pointer must be freed by the user to avoid memory leakage. See the FreeMemory task description in "Memory Model Utility".
	len (32)	O	Length of DLLP stored in memory in dwords.
	ei_code(32)	O	Error Injection code of DLLP sent. See "Data Link Layer Transmit DLLP Error Injection" for a list of the codes.
	receive_status (32)	O	Status of the received DLLP. See Include/ pciesvc_parms.v for bit definitions RECEIVED_DLLP_STATUS_*_BIT.
	valid (1)	O	Pointer is valid
SetReplayTimeout Replay timer limit will be automatically updated per spec Table "Unadjusted Replay Timer Limits" each time either link_width, freq_select, or MAX_PAYLOAD_SIZE_VAR changes unless this task is called.	value (32)	I	Length of the replay timer in symbols. If called, timeout value is sticky. Setting to value = 0 will enable automatic updates.
SetAttachedReplayTimeout Attached Replay timer limit will be automatically updated per spec, Table "Unadjusted Replay Timer Limits", each time either link_width, freq_select, or MAX_PAYLOAD_SIZE_VAR changes unless this task is called.	value (32)	I	Length of the attached replay timer in symbols. Used to check the DUT's replay timer. Setting value = 0 will enable automatic updates if MAX_PAYLOAD_SIZE_*VAR*, link_width, or speed change.

Table 5-3 User Specific – Link Interface (Continued)

Task Name	Arguments (length)	I/O	Values
SetAttachedReplayTimeoutTolerance Used to set tolerances for checking replay timer of DUT.	value (32)	I	Number of symbols in which retry may be received after a timeout. This should account for clock domain crossings, ACK processing variances, etc.
	link_width (32)	I	Link width value pertains to. Each link width can have a different value.
SetMaxAckNakLatency AckNak Latency timer will be automatically updated per spec, Table “Ack Transmit Latency Limit”, each time either link_width, freq_select, or MAX_PAYLOAD_SIZE_VAR changes unless this task is called.	value (32)	I	Max length of the ACK/NAK latency timer in symbols. If called, value is sticky. Setting to value = 0 will enable automatic updates.
SetMinAckNakLatency	value (32)	I	Min Length of the ACK/NAK latency timer in symbols. This value is not automatically updated, and defaults to 0.
SetMaxAttachedAckNakLatency Attached AckNak Latency timer will be automatically updated per spec, Table “Ack Transmit Latency Limit”, each time either link_width, freq_select, or MAX_PAYLOAD_SIZE_VAR changes unless this task is called.	value (32)	I	Max length of the attached ACK/NAK latency timer in symbols. Used to check DUT’s AckNak latency timer. If called, value is sticky. Setting value = 0 will enable automatic updates if MAX_PAYLOAD_SIZE_*VAR* or link_width change.
SetMinAttachedAckNakLatency Used to check DUT’s AckNak latency timer.	value (32)	I	Min length of the attached ACK/NAK latency timer in symbols. Default value = 0.
SetAttachedAckNakLatencyTolerance Used to set tolerances for checking AckNak latency timer of DUT.	value (32)	I	Variance in number of symbols in which ACK may be received. This should account for clock domain crossings, TLP processing variances, etc.
	link_width (32)	I	Link width value pertains to. Each link width can have a different value.
SetMaxAttachedNakLatency NAK latency limit will be automatically updated per spec, internal delay, each time either link_width, freq_select, or MAX_PAYLOAD_SIZE_VAR changes unless this task is called.	value (32)	I	Max NAK latency limit in symbols. Used to check DUT’s NAK latency. If called, value is sticky. Setting to value = 0 will enable automatic updates.

Table 5-3 User Specific – Link Interface (Continued)

Task Name	Arguments (length)	I/O	Values
SetAckFactor Used to update AckFactor table entries listed in spec. Values used to automatically set replay timer and AckNak latency values.	max_payload_size	I	Possible value, power of 2: 128, 256, 512, 1024, 2048, 4096.
	freq_select	I	
	link_width	I	Possible value: 1,2,4,8,12,16,32.
	value	I	
DisplayAttachedReplayTimeoutTolerances Displays replay timer checker tolerances for link widths of 1, 2,4,8,12,16, and 32.	n/a	n/a	Displays replay timeout tolerances used to check attached replay timers. Tolerances are per link_width.
DisplayAttachedAckNakLatencyTolerances Displays AckNak Latency checker tolerances for link widths of 1, 2,4,8,12,16, and 32.	n/a	n/a	Displays AckNak latency tolerances used to check attached AckNak latency timers. Tolerances are per link_width.
InitiateASPML0sEntry Directs VIP to enter ASPM Tx L0s. Call InitiateASPMExit task to direct VIP back to L0.	n/a	n/a	
InitiateASPML1Entry Directs VIP to enter ASPM L1. Call InitiateASPMExit task to direct VIP back to L0.	n/a	n/a	ENABLE_ASPM_L1_ENTRY must be enabled to enter L1.
InitiatePML1Entry Directs VIP to enter PM L1. Call InitiatePMExit task to direct VIP back to L0.	n/a	n/a	ENABLE_ASPM_L1_ENTRY must be enabled to enter L1.
InitiatePML23Entry Directs VIP to enter PM L2/L3. Call InitiatePMExit task to direct VIP back to L0.	n/a	n/a	
InitiatePMExit Directs VIP to transition back to L0 from PM low power state.	block	I	If block is set, task returns after low power state is exited. It not set, task returns immediately.
InitiateASPMExit Directs VIP to transition back to L0 from ASPM low power state.	block	I	If block is set, task returns after low power state is exited. It not set, task returns immediately.

5.4 Data Link Layer Parameters

Data Link Layer parameters are listed in Table 5-4.

Table 5-4 Data Link Layer parameters

Parameter Name	Settable VAR?*	Type	Range	Default	Description
PCIE_SPEC_VER					
	Yes	Real	1.1, 2.0, 2.1, 3.0	PCIE_ SPEC_ VER _2_1	See Include/pciesvc_parms.v: PCIE_SPEC_VER_* Note: Please set this in the model.
MAX_PAYLOAD_SIZE					
	Yes	Integer	2^x x = 7-12	128 bytes	Set replay timer and AckNak Latency timers per spec Tables *“Unadjusted Replay Timer Limits” and “Ack Transmit Latency Limit”*. Also used to check the DUT’s replay and AckNak Latency timers. Must match the Max_Payload_Size in DUT’s config register.
MAX_NUM_REPLAYS					
	Yes	Integer	3-5	4	Maximum number of replays before the link gives up on transmitting a TLP
MIN_NAK_LATENCY					
	Yes	Integer	0 - large value	0 ns	Minimum latency until a NAK is issued
MAX_NAK_LATENCY					
	Yes	Integer	0 - large value	30 ns	The maximum latency before a NAK is issued
MIN_INITFC_DELAY					
	Yes	Integer	0 - large value	0 ns	Minimum interval between InitFC DLLPs. Used to model delays to due media access. Value should be relatively small.
MAX_INITFC_DELAY					
	Yes	Integer	0 - large value	10 ns	Maximum interval between InitFC DLLPs. Used to model delays to due media access. Value should be relatively small.
MIN_UPDATEFC_DELAY					
	Yes	Integer	0 - large value	1	Minimum interval between UpdateFC DLLPs. Used to model delays to due media access. Value should be relatively small.
MAX_UPDATEFC_DELAY					
	Yes	Integer	0 - large value	100	Maximum interval between UpdateFC DLLPs. Used to model delays to due media access. Value should be relatively small.

Table 5-4 Data Link Layer parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
MIN_TX_IPG					
	Yes	Integer	0 - large value	0	Min gap between packets (TLPs and/or DLLPs) as measured in symbol times. 0 = packets can be packed in same symbol. 1 = packets will always start on lane 0 in symbol following END/EDB.
MAX_TX_IPG					
	Yes	Integer	0 - large value	1	Max gap between packets (TLPs and/or DLLPs) as measured in symbol times.
INITFC_TIMEOUT_NS					
	Yes	Integer	1us - large value	34us	Maximum time between receiving all 3 types (P, NP, CPL) of InitFC DLLPs. VIP will issue NOTICE if timeout value is exceeded.
UPDATEFC_TIMEOUT_NS					
	Yes	Integer	1us -large value	30us	Maximum time between receiving UpdateFC DLLPs. Timeout is tracked per VC/Type. VIP will issue NOTICE if timeout value is exceeded.
MAX_NUM_RETRY_BUFFER_DWORDS					
	No	Integer	16-2^16	4096	Maximum number of dwords the retry buffer can hold before it backpressures the Transaction Layer.
VC[0-7]_UPDATEFC_INTERVAL_NS					
	Yes	Integer	500 -large value	30,000	If credits for a VC have not been sent to the link partner for this interval, all 3 types of credits will be schedule for transmission. Setting to 0 disables scheduling credits. This should be set to 0 for most testing as periodic updates could mask credits being lost.
INITIAL_TRANSMIT_SEQUENCE_VALUE					
	Yes	Integer	0-4095	0	The first TLP will be transmitted with this value. Used to check rollover at 4096
INITIAL_RECEIVE_SEQUENCE_VALUE					
	Yes	Integer	0-4095	0	The first TLP sent must have this sequence number

Table 5-4 Data Link Layer parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
SENT_TLP_INTERFACE_MODE					
	Yes	Integer	See Description	1	Mode for queuing TLPs for SentTLP task. Bits are defined in Include/pciesvc_parms.v as SENT_TLP_INTERFACE_MODE_*_BIT
RECEIVED_TLP_INTERFACE_MODE					
	Yes	Integer	See Description	0	Mode for queuing TLPs for ReceivedTLP task. Bits are defined in Include/pciesvc_parms.v as RECEIVED_TLP_INTERFACE_MODE_*_BIT
SENT_DLLP_INTERFACE_MODE					
	Yes	Integer	See Description	0	Mode for queuing DLLPs for SentDLLP task. Bits are defined in Include/pciesvc_parms.v as SENT_DLLP_INTERFACE_MODE_*_BIT
RECEIVED_DLLP_INTERFACE_MODE					
	Yes	Integer	See Description	0	Mode for queuing DLLPs for ReceivedDLLP task. Bits are defined in Include/pciesvc_parms.v as RECEIVED_DLLP_INTERFACE_MODE_*_BIT
PERCENTAGE_TX_TLP_INSTEAD_OF_INITFC2					
	Yes	Integer	0-100	0	Enables sending a TLP to complete flow control initialization instead of sending and INITFC2 DLLP. A TLP must be queue in order for initialization to complete.
INTERNAL_DELAY_2_5G					
	Yes	Integer	1-128	10 ns	Internal delay used in "*Ack Transmit Latency Limit*" equation for replay timer and AckNak latency timer at 2.5G.
INTERNAL_DELAY_5G					
	Yes	Integer	1-128	10 ns	Internal delay used in "*Ack Transmit Latency Limit*" equation for replay timer and AckNak latency timer at 5G.
ATTACHED_INTERNAL_DELAY_2_5G					
	Yes	Integer	1-128	10 ns	Internal delay used in "*Ack Transmit Latency Limit*" equation to check attached replay timer and AckNak latency timer at 2.5G. Accounts for packet processing delay and transmission latency.

Table 5-4 Data Link Layer parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
ATTACHED_INTERNAL_DELAY_5G					
	Yes	Integer	1-128	10 ns	Internal delay used in “*Ack Transmit Latency Limit*” equation to check attached replay timer and AckNak latency timer at 5G. Accounts for packet processing delay and transmission latency.
MIN_TX_NULLIFIED_TLP_LEN					
	Yes	Integer	4-1024	4	Minimum nullified TLP length in dwords.
MAX_TX_NULLIFIED_TLP_LEN					
	Yes	Integer	4-1024	300	Maximum nullified TLP length in dwords.
TX_NULLIFIED_TLP_HDR0_VALUE					
	Yes	Integer	0 - 2^32	32’H7000_ 03FF	The default automatically generated nullified TLP is a MsgD, fatal error message. If the DUT forwards this TLP, it should be flagged by upper layers.
TX_NULLIFIED_TLP_HDR1_VALUE					
	Yes	Integer	0 - 2^32	32’HFFFF_ FF33	
TX_NULLIFIED_TLP_HDR2_VALUE					
	Yes	Integer	0 - 2^32	32’HBAAD_ BAAD	
TX_NULLIFIED_TLP_HDR3_VALUE					
	Yes	Integer	0 - 2^32	32’HDEAD_ BEEF	
ASPM_TIMEOUT_CNT_LIMIT					
	Yes	Integer	1000 - large value	20000	If DUT fails to respond to ASPM request handshake for this # symbols, NOTICE will be issued and ASPM entry will be aborted.
PM_TIMEOUT_CNT_LIMIT					
	Yes	Integer	1000 - large value	20000	If DUT fails to respond to PM request handshake for this # symbols, NOTICE will be issued and ASPM entry will be aborted.

Table 5-4 Data Link Layer parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
L0S_IDLE_TIMER_LIMIT_NS					
	Yes	Integer	10 - large value	0	When set to non-zero value, VIP will automatically enter ASPM L0s when transmitter is idle for this time. If set to 0, automatic ASPM L0s entry is disabled. For directed entry into L0s, use InitiateASPML0sEntry task.
ENABLE_ASPM_L1_ENTRY					
	Yes	Integer	0-1	0	This VAR must be set to 1 to enable ASPM L1 entry as requested by the VIP or the DUT.
ENABLE_ASPM_L1_1_ENTRY					
	Yes	Integer	0-1	0	This VAR must be set to 1 to enable ASPM L1_1 entry as requested by the VIP or the DUT.
ENABLE_ASPM_L1_2_ENTRY					
	Yes	Integer	0-1	0	This VAR must be set to 1 to enable ASPM L1_2 entry as requested by the VIP or the DUT.
ENABLE_PM_L1_1_ENTRY					
	Yes	Integer	0-1	0	This VAR must be set to 1 to enable PM L1_1 entry as requested by the VIP or the DUT.
ENABLE_PM_L1_2_ENTRY					
	Yes	Integer	0-1	0	This VAR must be set to 1 to enable PM L1_2 entry as requested by the VIP or the DUT.
LTR_L1_2_THRESHOLD_VALUE					
	Yes	Integer	0 - 2^16	0	Per spec L1_2 threshold value. ASPM L1_2 will be entered when snoop value*scale /no_snoop value*scale in received LTR msg are greater than this value*scale threshold. Applicable to Downstream port only.
LTR_L1_2_THRESHOLD_SCALE					
	Yes	Integer	0 - 2^16	0	Per spec L1_2 threshold scale. ASPM L1_2 will be entered when snoop value*scale /no_snoop value*scale in received LTR msg are greater than this value*scale threshold. Applicable to Downstream port only.
ENABLE_EI_TX_TLP_ON_RETRY					
	Yes	Integer	0-1	1	If enabled, EIs will be inserted on retries in addition to the first transmit attempt. Is not enabled, EI will be inserted only on the first transmit attempt.

Table 5-4 Data Link Layer parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
ENABLE_TRANSACTION_LOG					
	Yes	Integer	0-1	1	If enabled (and the transaction logger has been enabled via *msglog_control*, see below) transactions will be written to the transaction log.
ENABLE_TX_TLP_REPORTING					
	Yes	Integer	0-1	1	If enabled, transmitted TLPs will be reported to the global shadow memory. Make sure this is not enabled if the shadow memory is not instantiated.
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with "_VAR". This is useful if using higher level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.					

5.5 Data Link Layer Events

Types of Data Link Layer events are listed in Table 5-5.

Table 5-5 Data Link Layer events

Event Name	Description
event_sent_dllp	DLLP was sent.
event_sent_user_dllp	DLLP was sent via the user interface
event_received_dllp	DLLP was received. Can be used in conjunction with ReceivedDLLP task to indicate a DLLP is available.
event_sent_tlp	TLP was sent. Can be used in conjunction with SentTLP task to indicate a TLP is available.
event_received_tlp	TLP was received. Can be used in conjunction with ReceivedTLP task to indicate a TLP is available.
event_send_tlp	Event triggered when any TLP is sent.
event_send_dllp	Event triggered when any DLLP is sent.
event_receive_tlp	Event triggered when any TLP is received.

Table 5-5 Data Link Layer events (Continued)

Event Name	Description
event_receive_dllp	Event triggered when any DLLP is received
event_send_pm_active_state_nak	Event triggered when an PM_ACTIVE_STATE_NAK TLP needs to be sent to the link partner. The PM_ACTIVE_STATE_NAK TLP transmission will need to be initiated by the test case via the driver application or the TL user interface
event_rx_active_state_nak	Event triggered when an PM_ACTIVE_STATE_NAK TLP is received. If the enable_auto_aspm_l1_req_termination configuration variable is set, the DL will automatically terminate the PM handshake. If not set, the test case will need to terminate the PM handshake via InitiatePMExit() or by sending a TLP.

5.6 Data Link Layer ASCII Signals

ASCII signals on the Data Link Layer are listed in Table 5-6.

Table 5-6 Data Link Layer ASCII signals

Signal Name	Description
ascii_tx_tlp_type	Sent TLP type, defined by {fmt, type} fields.
ascii_tx_tlp_seq_num	Sent TLP sequence number.
ascii_tx_tlp_ei_code	TLP sent with this EI.
ascii_tx_dllp_type	Sent DLLP type.
ascii_tx_dllp_seq_num	Sent DLLP sequence number for ACK/NAK.
ascii_tx_dllp_credit_vc	Sent DLLP VC.
ascii_tx_dllp_credit_data_value	Sent DLLP data credit value.
ascii_tx_dllp_credit_hdr_value	Sent DLLP header credit value.
ascii_rx_tlp_type	Received TLP type, defined by {fmt, type} fields.
ascii_rx_tlp_seq_num	Received TLP sequence number.
ascii_rx_dllp_type	Received DLLP type.
ascii_rx_dllp_seq_num	Received DLLP sequence number for ACK/NAK.
ascii_rx_dllp_credit_vc	Received DLLP VC.
ascii_rx_dllp_credit_data_value	Received DLLP data credit value.
ascii_rx_dllp_credit_hdr_value	Received DLLP header credit value.
ascii_dlcmsm_state	Data Link Control Management State Machine.
ascii_vc[0-7]_fcsm_state	Flow Control State Machine for VC[0-7]
ascii_tx_dllp_ei_code	DLLP error codes.
ascii_aspm_state;	ASPM state

Table 5-6 Data Link Layer ASCII signals (Continued)

Signal Name	Description
ascii_pm_state;	PM state
ascii_tx_callback;	Callback being executed on TX side.
ascii_rx_callback;	Callback being executed on the RX side.
ascii_fc_init_state;	Flow control init state

5.7 Data Link Layer Statistics

Types of Data Link Layer statistics are listed in Table 5-7.

Table 5-7 Data Link Layer statistics

Stat Name	Width	Description
stat_num_tlp_sent	32	Number of TLPs sent by the link
stat_num_tlp_received	32	Number of TLPs received by the link
stat_num_good_tlp_sent	32	Number of TLPs sent without error
stat_num_good_tlp_received	32	Number of TLPs received without error
stat_num_bad_tlp_sent	32	Number of TLPs sent with error
stat_num_bad_tlp_received	32	Number of TLPs received with error
stat_num_good_retries_sent	32	Number of retry TLPs sent without error
stat_num_bad_retries_sent	32	Number of retry TLPs sent with error
stat_num_nullified_tlp_received	32	Number of nullified TLPs received without error
stat_num_dllp_sent	32	Number of DLLPs sent by the link
stat_num_dllp_received	32	Number of DLLPs received by the link
stat_num_good_dllp_sent	32	Number of DLLPs sent without error
stat_num_good_dllp_received	32	Number of DLLPs received without error
stat_num_bad_dllp_sent	32	Number of DLLPs sent with error
stat_num_bad_dllp_received	32	Number of DLLPs received with error
stat_num_retries_sent	32	Number of packets that had to be retransmitted
stat_num_phy_retrain	32	Number of times the link was retrained to due the replay attempt counter rolling over
stat_num_ack_sent	32	Number of ACK DLLPs sent
stat_num_ack_received	32	Number of ACK DLLPs received
stat_num_nak_sent	32	Number of NAK DLLPs sent
stat_num_nak_received	32	Number of NAK DLLPs received
stat_num_retries_due_to_replay_timeout	32	Number of retried packets due to replay timeout
stat_num_retries_due_to_nak	32	Number of retried packets due to a NAK
stat_num_duplicate_tlp_received	32	Number of received TLPs with the same sequence number

Table 5-7 Data Link Layer statistics (Continued)

Stat Name	Width	Description
stat_num_out_of_order_tlp_received	32	Number of TLPs received with out of order sequence numbers
stat_num_vc#_updatefc_received	32	Number of UpdateFC DLLPs received on VC#
stat_num_vc#_p_updatefc_received	32	Number of UpdateFC P DLLPs received on VC#
stat_num_vc#_np_updatefc_received	32	Number of UpdateFC NP DLLPs received on VC#
stat_num_vc#_cpl_updatefc_received	32	Number of UpdateFC CPL DLLPs received on VC#
stat_num_vc#_updatefc_sent	32	Number of UpdateFC DLLPs sent on VC#
stat_num_vc#_p_updatefc_sent	32	Number of UpdateFC P DLLPs sent on VC#
stat_num_vc#_np_updatefc_sent	32	Number of UpdateFC NP DLLPs sent on VC#
stat_num_vc# _cpl_updatefc_sent	32	Number of UpdateFC CPL DLLPs sent on VC#
stat_num_vc#_initfc1_received	32	Number of InitFC1 DLLPs received on VC#
stat_num_vc#_initfc1_p_received	32	Number of InitFC1 P DLLPs received on VC#
stat_num_vc#_initfc1_np_received	32	Number of InitFC1 NP DLLPs received on VC#
stat_num_vc#_initfc1_cpl_received	32	Number of InitFC1 CPL DLLPs received on VC#
stat_num_vc#_initfc1_sent	32	Number of InitFC1 DLLPs sent on VC#
stat_num_vc#_initfc1_p_sent	32	Number of InitFC1 P DLLPs sent on VC#
stat_num_vc#_initfc1_np_sent	32	Number of InitFC1 NP DLLPs sent on VC#
stat_num_vc#_initfc1_cpl_sent	32	Number of InitFC1 CPL DLLPs sent on VC#
stat_num_vc#_initfc2_received	32	Number of InitFC2 DLLPs received on VC#
stat_num_vc#_initfc2_p_received	32	Number of InitFC2 P DLLPs received on VC#
stat_num_vc#_initfc2_np_received	32	Number of InitFC2 NP DLLPs received on VC#
stat_num_vc#_initfc2_cpl_received	32	Number of InitFC2 CPL DLLPs received on VC#
stat_num_vc#_initfc2_sent	32	Number of InitFC2 DLLPs sent on VC#
stat_num_vc#_initfc2_p_sent	32	Number of InitFC2 P DLLPs sent on VC#
stat_num_vc#_initfc2_np_sent	32	Number of InitFC2 NP DLLPs sent on VC#
stat_num_vc#_initfc2_cpl_sent	32	Number of InitFC2 CPL DLLPs sent on VC#
stat_num_vendor_dllp_received	32	Number of vendor specific DLLPs received
stat_num_vendor_dllp_sent	32	Number of vendor specific DLLPS sent
stat_num_phy_link_down_events	32	Number of times the LTSSM reported link down dropping
stat_num_ei_tx_tlp_corrupt_lcrc	32	Number of times the corrupt LCRC EI was injected
stat_num_ei_tx_tlp_illegal_seq_number	32	Number of times the illegal sequence number EI was injected. Illegal sequence number is: (seq_num + 1) % 4096 thru (seq_num + 2047) % 4096

Table 5-7 Data Link Layer statistics (Continued)

Stat Name	Width	Description
stat_num_ei_tx_tlp_duplicate_seq_number	32	Number of times the duplicate sequence number EI was injected. Duplicate sequence number is: (seq_num + 2048) % 4096 thru (seq_num + 4095) % 4096
stat_num_ei_tx_tlp_nullified	32	Number of times the nullify TLP EI was injected
stat_num_ei_tx_tlp_nullified_good_lcrc	32	Number of times the nullify good TLP EI was injected
stat_num_ei_tx_tlp_nullified_corrupt_lcrc	32	Number of times the nullify corrupt lcrc TLP EI was injected
stat_num_ei_tx_tlp_corrupt_disparity	32	Number of times the corrupt disparity EI was injected during a TLP
stat_num_ei_tx_tlp_missing_start	32	Number of times the missing start delimiter EI was injected during a TLP
stat_num_ei_tx_tlp_missing_end	32	Number of times the missing end delimiter EI was injected during a TLP
stat_num_ei_tx_tlp_bit_flip	32	Number of times the bit flip EI was injected during a TLP
stat_num_ei_tx_tlp_code_violation	32	Number of times the code violation EI was injected during a TLP
stat_num_ei_tx_tlp_scrambler_error	32	Number of times the scrambler error EI was injected during a TLP
stat_num_ei_tx_tlp_8g_corrupt_header_crc	32	Number of times the corrupt STP token header FCRC EI was injected during a TLP in 8G mode.
stat_num_ei_tx_tlp_8g_corrupt_header_parity	32	Number of times the corrupt STP token header parity EI was injected during a TLP in 8G mode.
stat_num_ei_tx_dllp_corrupt_crc	32	Number of times the transmit DLLP corrupt CRC EI was injected
stat_num_ei_tx_dllp_unknown_type	32	Number of times the unknown DLLP type was injected
stat_num_ei_tx_dllp_rsvd_non_zero	32	Number of times the EI where reserved fields are nonzero was injected
stat_num_ei_tx_dllp_duplicate_ack	32	Number of times the EI duplicate ACKs was injected
stat_num_ei_tx_dllp_corrupt_disparity	32	Number of times the corrupt disparity EI was injected during a DLLP
stat_num_ei_tx_dllp_bit_flip	32	Number of times the bit flip EI was injected during a DLLP
stat_num_ei_tx_dllp_missing_start	32	Number of times the missing start delimiter EI was injected during a DLLP
stat_num_ei_tx_dllp_missing_end	32	Number of times the missing end delimiter EI was injected during a DLLP
stat_num_ei_tx_dllp_code_violation	32	Number of times the code violation EI was injected during a DLLP
stat_num_ei_tx_dllp_scrambler_error	32	Number of times the scrambler error EI was injected during a DLLP

Table 5-7 Data Link Layer statistics (Continued)

Stat Name	Width	Description
stat_num_ei_tx_dllp_8g_corrupt_header	32	Number of times the corrupt SDP token header EI was injected during a DLLP in 8G mode.
stat_num_ei_rx_tlp_withhold_ack_nak	32	Number of times ACK/NAK was withheld from transmission following reception of a TLP
stat_num_ei_rx_tlp_nak_good_tlp	32	Number of times the NAK good TLPs EI was injected
stat_num_ei_tx_replay_count_failure	32	Number of times the replay count failure EI was injected
stat_num_ei_rx_ignore_dllp_ack	32	Number of times the sent ACK NAK suppress EI was injected
stat_num_ei_tx_alignment_stp_wrong_lane	32	Number of times the alignment 2 STP wrong lane EI was injected
stat_num_ei_tx_alignment_sdp_wrong_lane	32	Number of times the alignment 2 SDP wrong lane EI was injected
stat_num_ei_tx_alignment_2_stp_per_symbol	32	Number of times the alignment 2 STP per symbol time EI was injected
stat_num_ei_tx_alignment_2_sdp_per_symbol	32	Number of times the alignment 2 SDP per symbol time EI was injected
stat_num_ei_tx_alignment_stp_packet_gap	32	Number of times the alignment STP packet gap EI was injected
stat_num_ei_tx_alignment_sdp_packet_gap	32	Number of times the alignment SDP packet gap EI was injected

5.8 Data Link LayerTransmit TLP Error Injection

Data Link Layer transmit TLP error injection codes are listed in Table 5-8.

Table 5-8 Data Link Layer transmit TLP error injection codes

Name	Type	Range	Description	DUT Action
EI_TX_TLP_PERCENTAGE				
	Integer	0-100	Percentage probability of a TLP type EI occurring.	
EI_TX_TLP_CORRUPT_LCRC_WEIGHT				
	Integer	Typically 0-1000	Corrupts LCRC of TLP transmitted to the Phy. The LCRC is corrupted by adding 1 to the correct value.	DUT will NAK, then svc will retry
EI_TX_TLP_ILLEGAL_SEQ_NUM_WEIGHT				

Table 5-8 Data Link Layer transmit TLP error injection codes (Continued)

Name	Type	Range	Description	DUT Action
	Integer	Typically 0-1000	Send packet sequence number (seq_num + 1) % 4096 thru (seq_num + 2047) % 4096. The same TLP is sent twice: 1st with the Illegal sequence number, 2nd with correct sequence number. The 2nd good TLP is not sent until the DUT responds with the correct NAK.	The TLP should be discarded. The sequence number is NOT one of the 2048 previous sequence numbers (mod 4096), so the TLP is out of sequence. The DUT should send a NAK.
EI_TX_TLP_DUPLICATE_SEQ_NUM_WEIGHT				
	Integer	Typically 0-1000	Send packet with sequence number (seq_num + 2048) % 4096 thru (seq_num + 4095) % 4096. The same TLP is sent twice: 1st with the duplicate sequence number, 2nd with correct sequence number. The DUT should respond with the correct ACK. If not additional packets are sent, a duplicate ACK will be received. Otherwise, the duplicate ACK could get aggregated with ACK from subsequent packet.	The TLP should be discarded. The DUT should send ACK since the TLP is a duplicate, i.e. the sequence number is one of the 2048 previous sequence numbers (mod 4096).
EI_TX_TLP_NULLIFIED_WEIGHT				
	Integer	Typically 0-1000	Data Link will send TLP to PHY with nullify indication. This EI will only generate nullified TLPs if TLPs are available for Tx. The nullified TLP header is programmed via TX_NULLIFIED_TLP_HDR0-3_VALUE parameter. The length of the nullified TLP is controlled via MIN/ MAX_TX_NULLIFIED_TLP_LEN. The default nullified TLP is an Error Msg with Fatal Msg code. The same sequence number is used by the nullified TLP and the subsequent good TLP.	DUT will ignore nullified TLP, and should not ACK/NAK. Since the sequence number will not be advanced, the DUT should use and ACK the next TLP with the same seq id.
EI_TX_TLP_NULLIFIED_GOOD_LCRC_WEIGHT				
	Integer	Typically 0-1000	Data Link will send TLP to PHY with nullify indication and good LCRC (not inverted as a correct nullify TLP). This EI will only generate nullified TLPs if TLPs are available for Tx. The nullified TLP is generated same as above, but the LCRC is not inverted.	DUT will not ignore nullified TLP since LCRC was not inverted as expected, and should send NAK. Since the sequence number will not be advanced, the DUT should ACK the last valid TLP sent.
EI_TX_TLP_NULLIFIED_CORRUPT_LCRC_WEIGHT				
	Integer	Typically 0-1000	Data Link will send TLP to PHY with nullified indication and corrupt the LCRC. This EI will only generate nullified TLPs if TLPs are available for Tx. The nullified TLP is generated same as above, but the LCRC is corrupted via adding 1 to correct value.	DUT will not ignore nullified TLP since LCRC did not pass, and should send NAK. Since the sequence number will not be advanced, the DUT should ACK the last valid TLP sent.
EI_TX_TLP_BIT_FLIP_WEIGHT				
	Integer	Typically 0-1000	Data Link will send EI code to Phy which will flip a bit in TLP at Byte indicated by Data Link.	DUT will NAK, then svc will retry. NOTE: The VIP cannot predict if the error will be detected inside the DLLP.
EI_TX_TLP_MISSING_START_WEIGHT				

Table 5-8 Data Link Layer transmit TLP error injection codes (Continued)

Name	Type	Range	Description	DUT Action
	Integer	Typically 0-1000	The Start delimiter, STP, will be replaced with idle or corrupted via disparity error, bit flip, or codeword violation.	The DUT should ignore the packet as it could not determine that a TLP was being received.
EI_TX_TLP_MISSING_END_WEIGHT				
	Integer	Typically 0-1000	The End delimiter, END, will be replaced with idle or corrupted via disparity error, bit flip, or codeword violation.	The DUT should NAK the packet as a TLP was being received. VIP will retry.
EI_TX_TLP_5G_CORRUPT_DISPARITY_WEIGHT				
	Integer	Typically 0-1000	Data Link will send EI code to Phy which will insert disparity error in TLP at Byte indicated by Data Link. Valid only for 2.5G and 5G modes.	DUT will NAK, then svc will retry. NOTE: The VIP can't predict if the error will be detected inside the DLLP.
EI_TX_TLP_5G_CODE_VIOLATION_WEIGHT				
	Integer	Typically 0-1000	Data Link will send EI code to Phy which will insert code violation in TLP at Byte indicated by Data Link. Valid only for 2.5G and 5G modes.	DUT will NAK, then svc will retry
EI_TX_TLP_SCRAMBLER_ERROR_WEIGHT **** Deprecated: Do not use *****				
	Integer	Typically 0-1000	Data Link will send EI code to Phy which will insert a scrambler error in TLP at Byte indicated by Data Link.	DUT will NAK, then svc will retry
EI_TX_TLP_8G_CORRUPT_HEADER_CRC_WEIGHT				
	Integer	Typically 0-1000	Data Link will send EI code to Phy which will corrupt token header FCRC. Only valid in 8G mode.	DUT is expected to transition to Recovery.
EI_TX_TLP_8G_CORRUPT_HEADER_PARITY_WEIGHT				
	Integer	Typically 0-1000	Data Link will send EI code to Phy which will corrupt token header parity. Only valid in 8G mode.	DUT is expected to transition to Recovery.

5.9 Data Link Layer Transmit DLLP Error Injection

Data Link Layer transmit DLLP error injection codes are listed in Table 5-9.

Table 5-9 Data Link Layer transmit DLLP error injection codes

Name	Type	Range	Description	DUT Action
EI_TX_DLLP_PERCENTAGE				
	Integer	0-100	Percentage probability of a transmit DLLP type EI occurring.	

Table 5-9 Data Link Layer transmit DLLP error injection codes (Continued)

Name	Type	Range	Description	DUT Action
EI_TX_DLLP_UNKNOWN_TYPE_WEIGHT				
	Integer	Typically 0-1000	Set DLLP “type” field to an unknown value.	When DUT receives a DLLP with an unknown type field, the packet is discarded with no further action. DUT should not NAK.
EI_TX_DLLP_CORRUPT_CRC_WEIGHT				
	Integer	Typically 0-1000	Corrupt CRC of packet sent to the Phy.	DLLP is discarded. NOTE: Need to figure out if DUT used the corrupt DLLP or not (use TransmitUserDLLP interface task)
EI_TX_DLLP_DUPLICATE_ACK_WEIGHT				
	Integer	Typically 0-1000	Generate an ACK for a packet that has already been ACK’d. The sequence number in the DLLP must not be 2047 less than Next_rcv_seq.	Re-ACK a recently ACKed sequence id. DUT should ignore as long as the seq id is within the last 2048 ids.
EI_TX_DLLP_RSVD_NON_ZERO_WEIGHT				
	Integer	Typically 0-1000	Set reserved fields in DLLPs to non-zero value.	Reserved fields should be ignored, normal processing.
EI_TX_DLLP_BIT_FLIP_WEIGHT				
	Integer	Typically 0-1000	Data Link will send EI code to Phy which will flip a bit in DLLP at Byte indicated by Data Link.	If corrupting an ACK/NAK, DUT will retry the TLP if another ACK is not sent before retry begins. For all FC, PM, or Vendor DLLPs, DUT should sent the next TLP if available. NOTE: The VIP can’t predict if the error will be detected inside the DLLP.
EI_TX_DLLP_MISSING_START_WEIGHT				
	Integer	Typically 0-1000	The Start delimiter, SDP, will be replaced with idle or corrupted via disparity error, bit flip, or codeword violation.	The DUT should discard the DLLP.
EI_TX_DLLP_MISSING_END_WEIGHT				
	Integer	Typically 0-1000	The End delimiter, END, will be replaced with idle or corrupted via disparity error, bit flip, or codeword violation.	The DUT should discard the DLLP.
EI_TX_DLLP_5G_CORRUPT_DISPARITY_WEIGHT				
	Integer	Typically 0-1000	Data Link will send EI code to Phy which will insert disparity error in DLLP at Byte indicated by Data Link. Valid only for 2.5G and 5G modes.	If corrupting an ACK/NAK, DUT will retry the TLP if another ACK is not sent before retry begins. For all FC, PM, or Vendor DLLPs, DUT should sent the next TLP if available. NOTE: The VIP can’t predict if the error will be detected inside the DLLP.

Table 5-9 Data Link Layer transmit DLLP error injection codes (Continued)

Name	Type	Range	Description	DUT Action
EI_TX_DLLP_5G_CODE_VIOLATION_WEIGHT				
	Integer	Typically 0-1000	Data Link will send EI code to Phy which will insert code violation in DLLP at Byte indicated by Data Link. Valid only for 2.5G and 5G modes.	If corrupting an ACK/NAK, DUT will retry the TLP if another ACK is not sent before retry begins. For all FC, PM, or Vendor DLLPs, DUT should sent the next TLP if available.
EI_TX_DLLP_SCRAMBLER_ERROR_WEIGHT **** Deprecated: Do not use *****				
	Integer	Typically 0-1000	Data Link will send EI code to Phy which will insert a scrambler error in DLLP at Byte indicated by Data Link.	If corrupting an ACK/NAK, DUT will retry the TLP if another ACK is not sent before retry begins. For all FC, PM, or Vendor DLLPs, DUT should sent the next TLP if available.
EI_TX_DLLP_8G_CORRUPT_HEADER_WEIGHT				
	Integer	Typically 0-1000	Data Link will send EI code to Phy which will corrupt SDP Token header. Only valid in 8G mode.	If corrupting an ACK/NAK, DUT will retry the TLP if another ACK is not sent before retry begins. For all FC, PM, or Vendor DLLPs, DUT should sent the next TLP if available.

5.10 Data Link Layer Receive TLP Error Injection

Data Link Layer Receive TLP error injection codes are listed in Table 5-10.

Table 5-10 Data Link Layer Receive TLP error injection codes

Name	Type	Range	Description	DUT Action
EI_RX_TLP_PERCENTAGE				
	Integer	0-100	Percentage probability of a receive TLP type EI occurring.	
EI_RX_TLP_WITHHOLD_ACK_NAK_WEIGHT				

Table 5-10 Data Link Layer Receive TLP error injection codes (Continued)

Name	Type	Range	Description	DUT Action
	Integer	Typically 0-1000	VIP will not send an ACK or NAK to the link partner for a sent TLP.	Dut will retry after timeout (measure). Note that the only TLPs that the DUT should then send us should be between the last ACK actually sent and the ACK number scheduled but not sent. In other words. The DU is not allowed to send any NEW TLPs until replay is completed. Also note the order must be increasing during the replay.
EI_RX_TLP_NAK_GOOD_TLP_WEIGHT				
	Integer	Typically 0-1000	For TLP sent, indicate NAK needs to be sent to link partner regardless of TLP integrity.	Dut will retry

5.11 Data Link Layer Multiple Packet Error Injection

Data Link Layer multiple packet error injection codes are listed in Table 5-11.

Table 5-11 Data Link Layer multiple packet injection error codes

Name	Type	Range	Description	DUT Action
EI_RX_MULTI_TLP_PERCENTAGE				
	Integer	0-100	Percentage probability of a receive TLP type EI occurring on multiple packets.	
EI_RX_REPLAY_COUNT_FAILURE_WEIGHT				
	Integer	Typically 0-1000	Cause DUT replay_num to roll over by ignoring 4 consecutive TLPs. This will force DUT to initiate retraining.	Cause a replay count failure by either NAKing the same TLP 4 times, or causing an ACKNAK timeout 4 times on the same TLP. DUT should cause the link to retrain. After retraining, DUT should attempt to send TLP again.

5.12 Data Link Layer Packet Alignment Error Injection

Data Link Layer packet alignment error injection codes are listed in Table 5-12.

Table 5-12 Data Link Layer packet alignment error injection codes

Name	Type	Range	Description	DUT Action
EI_TX_ALIGNMENT_PERCENTAGE				
	Integer	0-100	Percentage probability of an alignment type EI occurring.	
EI_TX_ALIGNMENT_STP_WRONG_LANE_WEIGHT				
	Integer	Typically 0-1000	Data Link will start a TLP in lane other than 0 without a preceding END.	DUT should ignore TLP. Maybe even send NAK.
EI_TX_ALIGNMENT_SDP_WRONG_LANE_WEIGHT				
	Integer	Typically 0-1000	Data Link will start a DLLP in lane other than 0 without a preceding END.	DUT should ignore DLLP.
EI_TX_ALIGNMENT_TWO_STP_PER_SYMBOL_WEIGHT				
	Integer	Typically 0-1000	Data Link will start 2 TLPs in symbol time. Only valid at link widths of 32.	DUT should ignore the 2nd TLP.
EI_TX_ALIGNMENT_TWO_SDP_PER_SYMBOL_WEIGHT				
	Integer	Typically 0-1000	Data Link will start 2 DLLPs in symbol time. Only valid at link widths of 12 or more.	DUT should ignore the 2nd DLLP.
EI_TX_ALIGNMENT_STP_PACKET_GAP_WEIGHT				

Table 5-12 Data Link Layer packet alignment error injection codes (Continued)

Name	Type	Range	Description	DUT Action
	Integer	Typically 0-1000	Data Link will place a gap between END and STP which occur in the same symbol time.	DUT should ignore 2nd TLP.
EI_TX_ALIGNMENT_SDP_PACKET_GAP_WEIGHT				
	Integer	Typically 0-1000	Data Link will place a gap between END and SDP which occur in the same symbol time.	DUT should ignore 2nd DLLP.

5.13 Data Link Layer Internal Interface Tasks

Data Link Layer internal interface tasks are listed in Table 5-13.

Table 5-13 Transaction – Data Link Layer interface tasks

Task Name	Arguments	I/O	Values
TransmitTLP	tlp_ptr(32)	I	Pointer of the TLP packet to be sent.
	tlp_len(32)	I	Length of the data buffer containing the TLP in dwords.
	tlp_ei_code(32)	I	Error injection code
	req	I	Request from application layer to Transaction Layer
	ack	O	Acknowledgement that Transaction Layer has accepted the transmission request from the application.
ReceiveTLP	tlp_vh	I	VH the TLP was sent on. This MR FC_INIT state machine needs this info. Valid when Ack is asserted.
	tlp_vc	I	VC the TLP was sent on. This FC_INIT state machine needs this info. Valid when Ack is asserted. When MRIOV is enabled, this returns the VL value.
	tlp_ptr(32)	O	Pointer to the sent TLP.
	tlp_len(32)	O	Length of the sent TLP buffer in dwords.
	tlp_ei_code(32)	O	Error Injection code
	req	O	Request to from Transaction Layer to application layer to pass TLP contents.
	ack	I	Acknowledgement that TLP contents have been accepted by the application layer.

Table 5-13 Transaction – Data Link Layer interface tasks (Continued)

Task Name	Arguments	I/O	Values
TransmitCredits Instructs the link to send an UpdateFCP or InitFCP DLLP. InitFC1 will cause Flow Control Init state machine to start for that particular VC.	vh(32)	I	Virtual Hierarchy to send DLLP to.
	vc(32)	I	Virtual channel to send DLLP to.
	credit_type(32)	I	credit_type[1:0] == 2'b00 means NP credit_type[1:0] == 2'b01 means Posted credit_type[1:0] == 2'b10 means Completion type.
	hdr_credits(32)	I	Number of header credits sent
	data_credits(32)	I	Number of data credits sent
	credit_is_init	I	This is the first credits sent. Will be use in FC Initialization process.
	req(1)	I	Request to send credits
	ack(1)	O	Acknowledgement that request was accepted
ReceiveCredits Any UpdateFC and InitFC1 credits to forward to Transaction Layer. This task will be call on every clock cycle. Only 1 flow control DLLP will be sent per cycle.	vh(32)	O	Virtual Hierarchy Credits sent on.
	vc(32)	O	Virtual channel Credits sent on.
	credit_type(32)	O	credit_type[1:0] == 2'b00 means NP credit_type[1:0] == 2'b01 means Posted credit_type[1:0] == 2'b10 means Completion type.
	hdr_credits(32)	O	Number of header credits sent
	data_credits(32)	O	Number of data credits sent
	credit_is_init	O	This is the first credits sent. These are initialization credits.
	req(1)	O	Request to Transaction Layer
	ack(1)	I	Acknowledgement from Transaction Layer that request was accepted.

5.14 Data Link Layer to Physical Layer Interface Tasks

Data Link Layer to Physical Layer interface tasks are listed in Table 5-14.

Table 5-14 Data Link Layer – Physical Layer interface tasks

Task Name	Arguments	I/O	Values
TransmitPhy			
	data_out(256)	O	Link data, on per lane basis Lane 0 – data[7:0] Lane 1 – data[15:8] Lane 31 – data[248:255]
	control_bus(256)	O	Control_bus per lane: Control_bus[7:0] – lane 0. Control_bus[15:8] – lane 1. Control_bus[248:255] – lane 31. control_bus[lane_offset+0] - Indicates idle control_bus[lane_offset+1] - Indicates STP control_bus[lane_offset+2] - Indicated SDP control_bus[lane_offset+3] - Indicates EOF control_bus[lane_offset+4] - Indicates EDB control_bus[lane_offset+7:5] - Reserved
	ei_code(256)	O	ei_code per lane: ei_code[7:0] – lane 0. ei_code[15:8] – lane 1. ei_code[248:255] – lane 31.
	phy_hold(1)	I	Indication from Physical Layer to hold transmission
	phy_align_packet(1)	I	Indication from Phy that next packet must be aligned to lane 0. This will be used by the Phy create an opportunity between packets to insert skip set.
ReceivePhy	data_in(256)	I	One byte of outgoing data per lane.
	control_bus(256)	I	Indication of whether the outgoing byte is valid or not, or whether it is a start of TLP, end of TLP, etc. Each byte of the control bus corresponds to a byte of data.
	ei_code(256)	I	Error injection code. There is a 1 byte error injection code per byte of data.

6 Physical Layer

The Physical Layer contains the LTSSM, performs data scrambling and descrambling, and is responsible for lane deskew and interfacing with the link for transmit/receive data. There is a separate PIPE interface to each PCS.

6.1 Pysical Layer Module IOs

The module inputs/outputs for the Physical Layer are listed in Table 6-1.

Table 6-1 Physical Layer module I/Os

Name	I/O	Description
reset	I(1)	Active high reset. Must only be asserted for 100ns ONCE per simulation at the beginning.
pl_control	I(32)	Control signals into the phy for controlling things such as link retrain and entry/exit from low power states.
wake	I(1)	Internal wake signal, not user accessible. To wake/exit the model from L2 please see InitiatePMExit or InitiateASPMExit" in the DL User Tasks
link_width	O(32)	Width of the negotiated link
freq_select	O(32)	Frequency select for the transmit clock generator.
pl_status	O(32)	[0] = link_up. [1] = phy in recovery

6.2 Physical Layer Configuration Tasks

Tasks for configuring the Physical Layer are listed in Table 6-2.

Table 6-2 Tasks for configuring the Physical Layer

Task Name	Arguments (length)	I/O	Values
SetLinkWidth Sets the maximum link width the LTSSM is allowed to negotiate. When link training is bypassed, the maximum link width will be used. Calling this task also changes the supported link widths to all legal values below the max link width and sets the expected link width equal to the maximum link width.	link_width (32)	I	Set the link width that the LTSSM will try to negotiate. Use the integral values with SetLinkWidth(). For example: 1 for x1, 2 for x2, 16 for x16 etc., Note. Do note use the link width parameter names used by SetSupportedLinkWidthVector (see below).
SetSupportedLinkWidthVector Defined which link widths the LTSSM is allowed to negotiate to. An x1 link width must always be supported. No link widths above what was defined in SetLinkWidth are allowed.	link_width_vector_ i (32)	I	Supported Link Widths User should or together use the LINK_WIDTH_* parms defined in Include/ pciesvc_parms.v to create a vector.
SetExpectedLinkWidth Sets the expected negotiated link width. The link width is checked on the LTSSM state transition from Configuration.Complete to L0.	link_width_i (32)	I	Expected link width.
InitiateLinkWidthChange Directs the LTSSM to attempt to upconfigure/downconfigure the link width to the highest allowed link width in the supported link width vector field. Can only be called in L0. Calls to this task outside of L0 will be ignored.	n/a	n/a	n/a
SetLinkNumber Sets the link number that the LTSSM will advertise in TS1/TS2 ordered sets during link training. Default link number is 0.	link_number (32)	I	Link number assigned to the LTSSM.

Table 6-2 Tasks for configuring the Physical Layer (Continued)

Task Name	Arguments (length)	I/O	Values
SetSupportedSpeeds Instructs the LTSSM which speeds are supported during link training. If link training is bypassed, the highest supported speed is always used. If the VIP is a downstream port, calling this task will automatically set the target speed to the highest supported speed.	speeds (32)	I	Supported speeds. Parameters should be or'd together to create a vector. For example, (SPEED_2_5G | SPEED_5G | SPEED_8G) indicates support for 2.5G, 5G and 8G. The speed parms are located in Include/pciesvc_parms.v
SetTargetSpeed Sets the desired speed of operation for a downstream port. Target speed is only defined for root complex, and calling this task for all other devices will be flagged as a warning. The target speed field is automatically updated to the highest supported speed every time SetSupportedSpeedsVector is called.	target_speed (32)	I	Target speed operation. Users should or together the SPEED_* parms defined in Include/pciesvc_parms.v to set the supported speed.
SetExpectedSpeed Sets the expected speed of operation. The expected speed check is performed after the following two conditions have been satisfied: The LTSSM is in the L0 state. The downstream port has transmitted an SDP.	expected_speed_i (32)	I	Expected speed of operation. Users should use the SPEED_* parms defined in Include/pciesvc_parms.v to set the expected speed. Only one speed can be expected.
InitiateRetrainLink Causes the LTSSM to enter recovery and attempt to retrain the link. This task has same functionality as setting retrain_link bit in PCIE Cfg's Link Control Register.	n/a	n/a	n/a
InitiateSpeedChange Sets the directed_speed_change variable in the LTSSM, causing the LTSSM to enter recovery and attempt a speed change. Should only be called in L0. Calls to this task outside of L0 will be ignored.	n/a	n/a	n/a

Table 6-2 Tasks for configuring the Physical Layer (Continued)

Task Name	Arguments (length)	I/O	Values
ConfigurePipeWidthTable Sets the width of per lane data bus for a given speed. This task is valid for SPIPE PIPE 3 models only. Default operation is 8 bits for all speeds. Using this task on a PIPE 4 model will result in a NOTICE stating that the task is deprecated.	rate(2) width(2	I I	Data rate that is to be configured. Users must use the parameters PIPE_RATE_* defined in Verilog/Include/pciesvc_parms. The pipe data bus width that the SVC is supposed to operate at when at the above speed. Users must use the PIPE_DATA_BUS_WIDTH_* parameters defined in Verilog/Include/pciesvc_parms.v
ConfigurePipeWidthPclkTable Sets the width of per lane data bus and pope clock rate for a given link speed. This task is valid for MPIPE PIPE 4 models only. Default operation is eight bits for all speeds.	rate (2) width (2) pclk_rate (2)		Data rate that is to be configured. Users must use the parameters PIPE_RATE_* defined in Verilog/Include/pciesvc_parms. The pipe width that the VIP is supposed to operate at when at the above speed. Users must use the PIPE_ WIDTH_* parameters defined in Verilog/Include/pciesvc_parms.v. Note that PIPE_WIDTH_* is not the same as PIPE_DATA_BUS_WIDTH. They are different PIPE signals with different values. Frequency at which the pipe clock will operate. Takes in the parms PIPE_PCLK_RATE_* defined in Verilog/ Include/pciesvc_parms.v See Table 3-2 in the PIPE spec for all legal combinations of rate, with and pclk_rate.
SetPhyEnable Allows the user to force the LTSSM to send training sets with the disable link bit and then transition to the disabled state.	enable (1)	I	When set to a 1, the phy will allow the LTSSM to transition out of the disabled state. When set to 0, the LTSSM will be forced to transition to the disabled state.

Table 6-2 Tasks for configuring the Physical Layer (Continued)

Task Name	Arguments (length)	I/O	Values
SetHotResetMode This task allows the user to initiate a hot reset and/or force the LTSSM to stay in the hot reset state.	mode (32)	I	Task takes a parameter defined in PhyLayer/pciescc_phylayer_parms.v. HOT_RESET_INACTIVE – In this mode the LTSSM will not initiate a hot reset. If the LTSSM is in the hot_reset state then it will transition to a Detect state. HOT_RESET_FORCE – If not in the hot_reset state, the LTSSM will begin sending training sets with the hot reset bit set and transition to hot_reset. When in the hot _reset state, the LTSSM will remain in this state until the hot reset mode has been changed in inactive or the hot reset timer expires. HOT_RESET_WAIT – If not in the hot_reset state, the LTSSM will begin sending training sets with the hot reset bit set and transition to the hot reset state. If in the hot_reset state, the LTSSM will wait until it has sent 2 TS1 ordered sets with the hot reset bit set with lane and link number set to the appropriate values, then transition to Detect.
InitiateLoopback Instructs the VIP to enter loopback as a loopback master.	enable_i (1)	I	When set to 1, VIP will attempt to enter loopback mode as a loopback master. When cleared, the VIP will exit loopback mode if it was a loopback master.

Table 6-2 Tasks for configuring the Physical Layer (Continued)

Task Name	Arguments (length)	I/O	Values
SetEnterCompliance	value (1)	I	Sets the enter compliance bit of the link control register. This is used to force the attached link into Polling.Compliance, and is also optionally used when entering loopback.
SetLaneReversalMode Configures the VIP support of lane reversal.	mode (32)	I	The modes are: UNSUPPORTED – The VIP will not attempt a lane reversal if during the configuration states it is unable to form a link. SUPPORTED- The VIP will attempt a lane reversal if it is unable to form a link using normal lane ordering. FORCED – The VIP will reverse its lanes from 0-(n-1) to (n-1)-0, where N is the link width as set using the SetLinkWidth task. This mode is intended for testing purposed only. It allows a user to test DUT lane reversal support without having to change the wiring. When calling this task, users must use the LANE_REVERSAL* parms in Verilog/ Include/pciesvc_parms.v.

6.3 Physical Layer User Tasks

The following tasks can be used

Tasks that can be used to query the Physical Layer statusa re listed in Table 6-3.

Table 6-3 Physical Layer user tasks

Task Name	Arguments (length)	I/O	Values
GetConfiguredLinkWidth Returns the link width the VIP is configured at.	link_width (32)	O	Returns the maximum number of lanes the pl is configured to utilize.
GetNegotiatedLinkWidth Returns the current operating link width	link_wdith (32)	O	Current operating link width. This task should only be called when the link is up.

Table 6-3 Physical Layer user tasks (Continued)

Task Name	Arguments (length)	I/O	Values
GetNegotiatedSpeed Returns current operating speed	speed (32)	O	Returns the current operating speed. Values are: SPEED_2_5G SPEED_5_0G SPEED_8_0G parms are defined in Include/pciesvc_parms.v
GetLtssmState Returns the current LTSSM state.	ltssm_state (32)	O	The value returned is the current state of the LTSSM. LTSSM state parameters are defined in PhyLayer/pciesvc_phylayer_parms.v.
RequestHotPlugMode	input [31:0] hot_reset_mode	I	Causes the LTSSM to behave as if it has either been physically unplugged or plugged back in. The hotplug mode parameters are defined in Verilog/Include/pciesvc_parms.v. Modes are as follows: HOTPLUG_UNPLUG- The phy is unplugged and will not respond to the other side of the link until the hotplug mode is changed. HOTPLUG_WAIT- The phy is enabled but will not exit detect.quiet until the other side of the link has done so. HOTPLUG_MONITOR- The phy will check incoming training sets and make sure that the DUT is timing out correctly in detect/polling.active, but the phy will not exit detect in this state. HOTPLUG_DETECT- This is the normal state. The phy will go through the detect states per the spec.

Tasks that can be used to pass time until the LTSSM reaches a particular state are listed in Table 6-4.

Table 6-4 Tasks used to consume time waiting for a state

Task Name	Arguments (length)	I/O	Values
WaitForLtssmState Blocks until the LTSSM has reached the given state. If the LTSSM is already in the specified state then no action is taken.	ltssm_state_i (32)	I	Waits until the LTSSM has reached a particular state. LTSSM state parameters are defined in PhyLayer/pciesvc_phylayer_parms.v
WaitUntilLinkUp Blocks until link_up goes high. If the link is already up then there is no action.	N/A	N/A	N/A
WaitUntilLtssmStateChange Blocks until the LTSSM has exited the current state	N/A	N/A	N/A
ConfigureLane This task associates an advertised link number with a physical link. The default behavior is to associate physical lane 0 with advertised lane 0, physical lane 1 will advertise itself as lane 1 and so on. Example- ConfigureLane(0,1,1) will instruct the LTSSM have physical lane 0 advertise itself as lane 1 in all TS1/TS2 ordered sets. This is intended to be used only for LTSSM testing, the LTSSM will not support arbitrary lane N to N mapping.	phy_id (32)		Phy number that user wishes to configure
	lane_num (32)		Lane number the phy should be associated with.
	enabled (32)		Enable the association.
DisplayStats Displays all stats in Physical Layer.	None		Displays all stats in PL
ClearStats Clears all stats in Physica Layer.	None	I	Clear all stats in PL

6.4 LTSSM User Training Set Tasks

The Physical Layer has the capability for a user to define the entire contents of transmitted TS1/TS2 training sets on a per lane basis. Tasks used to define the contents of TS1/TS2 training sets are listed in Table 6-5.

Note that these tasks should only be used for directed tests , and should not be used during normal operation.

In order to send user training sets, SetUserTxTs1/SetUserTxTs2 must be called to define the training sets on the lanes that a user training set is to be transmitted on. Once the user training sets are defined, the task SetUserTxTs1Pattern/SetUserTxTs2Pattern is called to define a pattern of user TS1 to normal TS1.

Example

SetUserTxTs1Pattern(1’b1, 1, 2, 1, 1) will enable user TS1 patterns, and the LTSSM will transmit between 1 and 2 user TS1 followed by 1 normal TS1.

The parameters USER_TX_TS1_LANE_MASK and USER_TX_TS2_LANE_MASK can be used to apply the user training sets only to certain lanes.

Table 6-5 LTSSM user training set tasks

Task Name	Arguments	I/O	Values
SetUserTxTs1	lane_num_i(32)	I	Lane number that is to be configured.
Defines a user training set and optionally an error injection for a given lane.	ei_bytenum(32)	I	Byte number that EI is to be injected on
	ei_code(32)	I	EI code to be applied to the error'd symbol. Symbol 0 of the user training set. MSB is the control bit for all symbols
	symbol0(9)	I	Symbol 1 of the user training set
	symbol1(9)	I	Symbol 2 of the user training set
	symbol2(9)	I	Symbol 3 of the user training set
	symbol3(9)	I	Symbol 4 of the user training set
	symbol4(9)	I	Symbol 5 of the user training set
	symbol5(9)	I	Symbol 6 of the user training set
	symbol6(9)	I	Symbol 7 of the user training set
	symbol7(9)	I	Symbol 8 of the user training set
	symbol8(9)	I	Symbol 9 of the user training set
	symbol9(9)	I	Symbol 10 of the user training set
	symbol10(9)	I	Symbol 11 of the user training set
	symbol11(9)	I	Symbol 12 of the user training set
	symbol12(9)	I	Symbol 13 of the user training set
	symbol13(9)	I	Symbol 14 of the user training set
	symbol14(9)	I	Symbol 15 of the user training set
	symbol15(9)	I	Symbol 16 of the user training set

Table 6-5 LTSSM user training set tasks (Continued)

Task Name	Arguments	I/O	Values
SetUserTxTs1Pattern When enabled, the LTSSM will send a random burst of user training sets. The size of the burst is set by the min/max arguments. In between the burst of user training sets the LTSSM will send normal TS1. The number of normal TS1 is defined by the min/max spacing arguments.	enable_i(32) min_user_tx_ts1_burst_i max_user_tx_ts1_burst_i min_user_tx_ts1_spacing_i max_user_tx_ts1_spacing_i	I I I I I	‘1’ enables the training set pattern, ‘0’ disables it Minimum number of user TS1 sets in the user burst. Maximum number of user TS1 sets in the user burst. Minimum number of normal TS1 sets between the error burst. Minimum number of normal TS1 sets between the error burst.

Table 6-5 LTSSM user training set tasks (Continued)

Task Name	Arguments	I/O	Values
SetUserTxTs2 Defines a user training set and optionally an error injection for a given lane.	lane_num_i(32)	I	Lane number that is to be configured.
	ei_bytenum(32)	I	Byte number that EI is to be injected on
	ei_code(32)	I	EI code to be applied to the error'd symbol.
	symbol0(9)	I	Symbol 0 of the user training set. MSB is the control bit for all symbols.
	symbol1(9)	I	Symbol 1 of the user training set
	symbol2(9)	I	Symbol 2 of the user training set
	symbol3(9)	I	Symbol 3 of the user training set
	symbol4(9)	I	Symbol 4 of the user training set
	symbol5(9)	I	Symbol 5 of the user training set
	symbol6(9)	I	Symbol 6 of the user training set
	symbol7(9)	I	Symbol 7 of the user training set
	symbol8(9)	I	Symbol 8 of the user training set
	symbol9(9)	I	Symbol 9 of the user training set
	symbol10(9)	I	Symbol 10 of the user training set
	symbol11(9)	I	Symbol 11 of the user training set
	symbol12(9)	I	Symbol 12 of the user training set
	symbol13(9)	I	Symbol 13 of the user training set
	symbol14(9)	I	Symbol 14 of the user training set
	symbol15(9)	I	Symbol 15 of the user training set

Table 6-5 LTSSM user training set tasks (Continued)

Task Name	Arguments	I/O	Values
SetUserTxTs2Pattern	enable(32)	I	'1' enables the training set pattern, '0' disables it
When enabled, the LTSSM will send a random burst of user training sets. The size of the burst is set by the min/max arguments. In between the burst of user training sets the LTSSM will send normal TS2. The number of normal TS2 is defined by the min/max spacing arguments.	min_user_tx_ts2_burst	I	Minimum number of user TS2 sets in the user burst.
	max_user_tx_ts2_burst	I	Maximum number of user TS2 sets in the user burst.
	min_user_tx_ts2_spacing	I	Minimum number of normal TS2 sets between the error burst.
	max_user_tx_ts2_spacing	I	Minimum number of normal TS2 sets between the error burst.
GetTxTs	lane_num_i(32)	I	Lane number that is to be configured.
Returns the training set that is currently being transmitted on lane_num. This task is intended to help users build a training set where they wish to modify only a single field of an outgoing normal TS1/TS2	symbol0(9)	O	Symbol 0 of the user training set
	symbol1(9)	O	Symbol 1 of the user training set
	symbol2(9)	O	Symbol 2 of the user training set
	symbol3(9)	O	Symbol 3 of the user training set
	symbol4(9)	O	Symbol 4 of the user training set
	symbol5(9)	O	Symbol 5 of the user training set
	symbol6(9)	O	Symbol 6 of the user training set
	symbol7(9)	O	Symbol 7 of the user training set
	symbol8(9)	O	Symbol 8 of the user training set
	symbol9(9)	O	Symbol 9 of the user training set
	symbol10(9)	O	Symbol 10 of the user training set
	symbol11(9)	O	Symbol 11 of the user training set
	symbol12(9)	O	Symbol 12 of the user training set
	symbol13(9)	O	Symbol 13 of the user training set
	symbol14(9)	O	Symbol 14 of the user training set
	symbol15(9)	O	Symbol 15 of the user training set

6.5 Equalization Tasks

Equalization tasks are listed in Table 6-6.

Table 6-6 Equalization Tasks

Task Name	Arguments (length)	I/O	Values
SetAttachedFSLF Sets what values the SPIPE will drive on the LocalFS/LocalLF PIPE bus. This task should not be called if not using an SPIPE model.	lane_en(32)	I	Bit vector to enable FS/LF values.
	fs(32)	I	full swing value
	lf(32)	I	low frequency value
SetTxTS1FSLF Set FS and LF values advertised to the link partner during Phase 1. FS and LF values are used by SVC to check requested preset/coefficients.	lane_en(32)	I	Bit vector to enable FS/LF values.
	fs(32)	I	full swing value
	lf(32)	I	low frequency value
GetRxTS1FSLF Get FS and LF values received in training sets by the SVC on each lane. These values are received from the link partner in TS1 during Phase 1.	lane (32)	I	Lane to query
	fs(32)	O	full swing value
	lf(32)	O	low frequency value
	valid(1)	O	When asserted, FS and LF have been captured from the link partner.
GetTxTS1FSLF Get local FS and LF values advertised by the SVC. These values are sent to the link partner in TS1 during Equalization Phase 1.	lane (32)	I	Lane to query
	fs(32)	O	full swing value
	lf(32)	O	low frequency value
	valid(1)	O	When asserted, FS and LF have been captured from the link partner.
SetEQEvaluationDelay Set Min/Max time to evaluate attached transmitter setting changes. Default value min/max values are 100/ 1000ns	lane_en(32)	I	Bit vector enable preset applies to.
	min(32)	I	Minimum delay time to evaluate equalization.
	max(32)	I	Maximum delay time to evaluate equalization.
SetRxEQEvalDelay The amount of time in ns that an SPIPE model takes to respond to an RxEqEval request. SPIPE only. Default values: min 10, max 50 ns	lane_en(32)	I	
	min(32)	I	
	max(32)	I	
SetEQPresetValidationDelay Set min/max preset/coefficient time to either accept or reject a request. Default: 0 ns	lane_en(32)	I	Bit vector enable preset applies to.
	min(32)	I	Minimum delay time to validate preset/ coefficient request. Default:
	max(32)	I	Maximum delay time to validate preset/ coeffiecient request.

Table 6-6 Equalization Tasks (Continued)

Task Name	Arguments (length)	I/O	Values
QueueEQFigureMeritResponse Sets a Figure of Merit response from equalization evaluation. Response range is 0 – 255. If no responses are queued then the SVC will return a figure of merit value of 255.	lane_en(32)	I	Bit vector enable the figure of merit response applies to.
	min(32)	I	Minimum Figure of Merit response from equalization evaluation.
	max(32)	I	Maximum Figure of Merit response from equalization evaluation.
QueueEQDirectionChangeResponse Sets the direction change response from equalization evaluation. Default is no change on any coefficients if no responses are queued.	lane_en(32)		Bit vector enable response applies to.
	direction_change_response(32)		Sets the direction change response from a link evaluation request. Please refer to the PIPE spec for encoding of this signal. Bits 6-31 of this argument are reserved.
QueueEQTxRequestPresetCoeff Queue either Preset or Coeff Request. SVC sends this request in TS1 at 8G in Recovery.Equalization. As long as a request is queued, SVC will issue a new request when the previous request completes. Phase will complete if no more requests are queued and transition criteria have been met. Downstream port: in Phase 0 (as init preset for Upstream port, default = 15), Phase 3 as new preset/coeff request. Upstream port: In Phase 2 as new preset/coeff request.	lane_en(32)	I	Bit vector which enables preset. For example, a value of three would enable lanes zero and one.
	preset_precursor(32)	I	Preset if preset_valid = 1 or precursor coefficient if preset_valid=0.
	preset_valid(1)	I	If asserted, preset entry is queued, else coeff will be queued and requested in appropriate phase.
	preshoot(32)	I	Same as precursor
	cursor_coeff	I	This is the Cursor Coefficient value.
	deemphasis(32)	I	Same as post cursor
	expect_reject(1)	I	When set to ‘b1 the SVC will expect the request to be rejected. When set to ‘b0 the SVC will expect the request to be accepted.
SetEQDownstreamPresetDefault Set per lane Downstream_Preset_Reg_Var to default value. Will be sent in EQTS1 by Downstream port in Phase 1. Defaults to 15.	lane_en(32)	I	Bit vector enable preset applies to.
	preset(32)	I	preset value

Table 6-6 Equalization Tasks (Continued)

Task Name	Arguments (length)	I/O	Values
SetEQPreset2CoeffTable This table is used to map received preset requests to coeffiecients for use in local Transmitter settings. Default behavior is to enable all presets at initial time and set each entry to the parameter values PRECURSOR_COEFFICIENT, CURSOR_COEFFICIENT and POSTCURSOR_COEFFICIENT.	preset(32)	I	preset value
	precursor_coeff(32)	I	precursor coefficient
	cursor_coeff(32)	I	cursor coefficient
	postcursor_coeff(32)	I	postcursor coefficient
	enable(1)	I	Enable preset entry. If the entry is not enabled then the SVC will reject any requests to use this preset.
SetEQExpectedPreset2CoeffTable This table is used to verify DUT's preset to coefficient mapping. Table should be programmed exactly like the DUT's preset table. If an entry is not enabled then SVC will not check received coefficients.	preset(32)	I	preset value
	precursor(32)	I	precursor coefficient corresponding to the preset value
	cursor(32)	I	cursor coefficient corresponding to the preset value
	postcursor(32)	I	postcursor coefficient corresponding to the preset value
	reject(1)	I	When set to 1 the SVC will expect for the preset to be rejected. Else the preset should be accepted.
	enable(1)	I	Enable preset entry. If enabled then the SVC will check that incoming TS1 show the correct coefficient values when a particular preset has been requested.
GetEQRxRequestedPresetCoefficients The SVC will capture preset and/or coefficients requested by the DUT. If "use_preset" field was set in TS OS, preset_valid will be asserted.	lane(32)	I	Lane number
	preset(32)	O	preset value
	use_preset(1)	O	use_preset bit
	precursor_coeff(32)	O	precursor coefficient
	cursor_coeff(32)	O	cursor coefficient
	postcursor_coeff(32)	O	postcursor coefficient
	status(32)	O	[0] : Request was processed [1]: Request was rejected
	valid(1)	O	If 1, preset and coefficients have been captured from the DUT but not accepted or rejected.

Table 6-6 Equalization Tasks (Continued)

Task Name	Arguments (length)	I/O	Values
SetEQUpstreamReceiverPresetHint Downstream Ports: Set the receiver preset hint transmitted in EQ TS2s to the DUT before equalization phase 0. Upstream Ports: No effect	lane_en(32)	I	Bit vector enable preset applies to.
	preset(32)	O	preset value
GetEQUpstreamReceiverPresetHint Downstream Ports: Gets the suggested receiver preset hint advertised in EQTS2 before equalization phase 0 Upstream ports: Gets the captured receiver preset hint received in EQ TS2s from the DUT before equalization phase 0. Upstream port only.	lane (32)	I	Lane number
	preset(32)	O	preset value
	valid(1)	O	If 1, receiver preset hint has been captured from the DUT.
SetEQDownstreamReceiverPresetHint Downstream Lanes: Sets the value of the downstream_receiver_preset_reg which controls the receiver preset hint advertised in EQTS1. *Upstream lanes: no effect*	lane_en (32)	I	32 bit lane enable vector
	preset_hint(32)	I	Preset hint value
GetEQDownstreamReceiverPresetHint Downstream lanes: Gets the value of the downstream_receiver_preset_reg which controls the receiver preset hint advertised in EQTS1. Upstream lanes: Gets the preset hint value received in EQTS1.	lane(32)	I	Lane to query
	preset_hint(32)	I	Preset hint value
SetPerformEqualization When 8G supported, this task is called, and then RetrainLink is called, Downstream port will initiate equalization. Valid for Down-stream port only.	N/A	N/a	Task sets the perform_equalization variable in the VIP. Note: VIP must not be running at 8G for equalization to be performed again. At 8g, use SetRequestEqualization to rerun.

Table 6-6 Equalization Tasks (Continued)

Task Name	Arguments (length)	I/O	Values
SetRequestEqualization Request equalization to be run again. This task sets TS2 symbol 6, bit 7 in Recovery.RcvrCfg @8G. See Spec Table 4-6.	N/A	N/A	Task sets the request_equalization variable in the VIP. Equalization will only be rerun if currently operating at 8G.
SetQiesceGuarantee			Task sets the quiesce_guarantee variable in the VIP. This variable is reflected in Symbol 6 bit 6 of TS2 OS transmitted at 8G in Recovery.RcvrCfg.
GetEqualizationStatus	status (32)	0	Get equalization status values: [0] = eq_complete [1] = phase 1 complete [2] = phase 2 complete [3] = phase 3 complete

6.6 Physical Layer Parameters

6.6.1 General Parameters

General Physical Layer parameters are listed in Table 6-7.

Table 6-7 Physical Layer general parameters

Parameter Name	Settable VAR?*	Type	Range	Default	Description
PCIE_SPEC_VER					
	Yes	Real	1.1, 2.0, 2.1, 3.0	PCIE_SPEC_VER_2_1	See Include/pciesvc_parms.v: PCIE_SPEC_VER_* Note: Please set this in the model.
PIPE_SPEC_VER					
	Yes	Integer	2, 4	PIPE_SPEC_VER_2	See Include/pciesvc_parms.v: PIPE_SPEC_VER_*
INVERT_TX_POLARITY					
	Yes	Integer	0-1	32'b0	When a bit is set, the corresponding lane will invert polarity on all outgoing data.
LOOPBACK_MASTER_SPEED_CHANGE_TIMEOUT_NS					
	Yes	Integer		1000	Speed change delay for loopback master.

Table 6-7 Physical Layer general parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
LOOPBACK_SLAVE_SPEED_CHANGE_TIMEOUT_NS					
	Yes	integer		2000	Speed change delay for loopback slave
LOOPBACK_EXIT_ELEC_IDLE_TIMEOUT_NS					
	Yes	integer		2000	Amount of time the LTSSM transmit electrical idle in Loopback.exit before entering detect.
LOOPBACK_MASTER_ENTER_COMPLIANCE_TX_TS1_TIMEOUT_NS					
	Yes	integer		2000	Time before the loopback master enters loopback.active if no TS1s are received w/ the enter compliance bit set
LOOPBACK_MASTER_NO_ENTER_COMPLIANCE_TX_TS1_TIMEOUT_NS					
	Yes	integer		20000	Time before the loopback master enters detect if no TS1s are received w/ the enter compliance bit clear;
MAX_TX_SKP_INTERVAL_IN_SYMBOL_TIMES					
	Yes	Integer	32 - large value	1538	Maximum number of symbol times before the upper phy will schedule the insertion of a SKP ordered set (2.5GT/s and 5 GT/s)
MIN_TX_SKP_INTERVAL_IN_BLOCKS					
	Yes	Integer	2 - large value	375	Minimum number of blocks before the upper phy must schedule the insertion of a SKP ordered set (8GT/s)
MAX_TX_SKP_INTERVAL_IN_BLOCKS					
	Yes	Integer	2 - large value	375	Maximum number of blocks before the upper phy must schedule the insertion of a SKP ordered set (8GT/s)
MIN_RX_SKP_INTERVAL_IN_SYMBOL_TIMES					
	Yes	Integer	32 - large value	1180	Minimum number of symbol times before the upper phy flag an error due to the lack of a SKP ordered set (2.5GT/s and 5 GT/s)
MAX_RX_SKP_INTERVAL_IN_SYMBOL_TIMES					
	Yes	Integer	32 - large value	1538	Maximum number of symbol times before the upper phy will flag an error due to the lack of a SKP ordered set (2.5GT/s and 5 GT/s)
MIN_RX_SKP_INTERVAL_IN_BLOCKS					
	Yes	Integer	2 - large value	375	Minimum number of blocks before the upper phy flags an error due to the lack of a SKP ordered set (8GT/s)
MAX_RX_SKP_INTERVAL_IN_BLOCKS					

Table 6-7 Physical Layer general parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
	Yes	Integer	2 - large value	375	Maximum number of blocks before the upper phy flags an error due to lack of a SKP ordered set (8GT/s)
MIN_TX_SKP_SYMBOLS_IN_ORDERED_SET					
	Yes	Integer	1-5	3	Minimum number of SKP symbols in an ordered set at 2.5GT/s and 5GT/s.
MAX_TX_SKP_SYMBOLS_IN_ORDERED_SET					
	Yes	Integer	1-5	3	Maximum number of SKP symbols in an ordered set at 2.5GT/s and 5GT/s.
MIN_TX_SKP_SYMBOLS_IN_ORDERED_SET_8G					
	Yes	Integer	1-5	3	Minimum number of SKP symbols in an ordered set at 8GT/s. Acceptable values: 1 – 5.
MAX_TX_SKP_SYMBOLS_IN_ORDERED_SET_8G					
	Yes	Integer	1-5	3	Maximum number of SKP symbols in an ordered set at 8GT/s. Acceptable values: 1 – 5.
DISABLE_SCRAMBLING					
	Yes	Integer	0-1	0	When set to '1' scrambling will be disabled, and the LTSSM will set the disable scrambling bit on all transmitted TS1 and TS2 ordered sets.
MIN_TX_LANE_SKEW_2_5G					
	Yes	Integer	0-16	0	Sets the minimum lane skew between transmit lanes in symbol times at 2.5Gb/s. Skew is randomized on every link down event, speed change, and at initial startup.
MAX_TX_LANE_SKEW_2_5G					
	Yes	Integer	0-16	0	Sets the maximum lane skew between transmit lanes in symbol times at 2.5Gb/s. Skew is randomized on every link down event, speed change, and at initial startup.
MIN_TX_LANE_SKEW_5G					
	Yes	Integer	0-16	0	Sets the minimum lane skew between transmit lanes in symbol times at 5Gb/s. Skew is randomized on every link down event, speed change, and at initial startup.
MAX_TX_LANE_SKEW_5G					
	Yes	Integer	0-16	0	Sets the maximum lane skew between transmit lanes in symbol times at 5Gb/s. Skew is randomized on every link down event, speed change, and at initial startup.
MIN_TX_LANE_SKEW_8G					

Table 6-7 Physical Layer general parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
	Yes	Integer	0-16	0	Sets the minimum lane skew between transmit lanes in symbol times at 8Gb/s. Skew is randomized on every link down event, speed change, and at initial startup.
MAX_TX_LANE_SKEW_8G					
	Yes	Integer	0-16	0	Sets the maximum lane skew between transmit lanes in symbol times at 8Gb/s. Skew is randomized on every link down event, speed change, and at initial startup.
MAX_RX_LANE_SKEW_2_5G					
	Yes	Integer	0-16	1	Maximum allowed lane skew before the VIP flags a skew violation on the receive side at 2.5G.
MAX_RX_LANE_SKEW_5G					
	Yes	Integer	0-16	1	Maximum allowed lane skew before the VIP flags a skew violation on the receive side at 5G.
MAX_RX_LANE_SKEW_8G					
	Yes	Integer	0-16	1	Maximum allowed lane skew before the VIP flags a skew violation on the receive side at 8G.
ENABLE_RELAXED_SKP_CHECKING					
	Yes	Integer	0-1	0	When set, the Physical Layer will relax its checking for the number of SKP symbols in a SKP ordered set. This should only be set when the VIP is connected to a switch or repeater that does not regenerate refclk.
MIN_SPIPE_PHYSTATUS_DELAY					
	Yes	Integer	0-1024	12 ns	Minimum number of pipe clk cycles the VIP will wait before asserting phystatus indicating completion of a phy function (e.g. power state change or rate change). Valid only for SPIPE models.
MAX_SPIPE_PHYSTATUS_DELAY					
	Yes	Integer	0-1024	16 ns	Maximum number of pipe clk cycles the VIP will wait before asserting phystatus indicating completion of a phy function (e.g. power state change or rate change). Valid only for SPIPE models.
MIN_SPIPE_PRESET_COEFFICIENTS_DELAY					
	Yes	Integer		8 ns	Minimum number of pipe clk cycles the SVC will wait before asserting local_tx_coefficients_valid when requested to get local preset coefficients. Valid only for SPIPE 8G models.
MAX_SPIPE_PRESET_COEFFICIENTS_DELAY					

Table 6-7 Physical Layer general parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
	Yes	Integer	0-1024	12 ns	Maximum number of pipe clk cycles the SVC will wait before asserting local_tx_coefficients_valid when requested to get local preset coefficients. Valid only for SPIPE 8G models.
GET_LOCAL_PRESET_COEFFICIENTS_TIMEOUT_NS					
	Yes	Integer	0-1024	128	Number of ns before an MPIPE flags a timeout on a get local preset coefficient request.
TX_SWING_2_5G					
	Yes	Integer	0-1024	0	Value driven on pipe signal tx_swing at 2.5G. Valid for mpipe 8G models only.
TX_SWING_5G					
	Yes		0-1024	0	Value driven on pipe signal tx_swing at 5G. Valid for mpipe 8G models only.
TX_SWING_8G					
	Yes		0-1024	0	Value driven on pipe signal tx_swing at 8G. Valid for mpipe 8G models only.
TX_MARGIN_2_5G					
	Yes		0-1024	0	Value driven on pipe signal tx_margin at 2.5G. Valid for mpipe 8G models only.
TX_MARGIN_5G					
	Yes		0-1024	0	Value driven on pipe signal tx_margin at 5G. Valid for mpipe 8G models only.
TX_MARGIN_8G					
	Yes		0-1024	0	Value driven on pipe signal tx_margin at 8G. Valid for mpipe 8G models only.
ASSERT_CLKREQ_N					
	Yes	Integer	0-1	0	When PCIE_SPEC_VER is set to 3.0, this var can be used to force assertion of CLKREQ#. When not asserted, model will drive CLKREQ# signal to 1 since it is declared as tri1 internally.
DISPLAY_NAME					
	No	String		pciesvc_pl0	Default display name for the Physical Layer.

Table 6-7 Physical Layer general parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
ENABLE_SYMBOL_LOG					
	Yes	Integer	0-1	0	If enabled and if the symbol log has not been disabled via msglog_control (see "Symbol Logging") symbols will be written to the symbol log.
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with "_VAR". This is useful if using higher level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.					

6.6.2 Physical Layer LTSSM-specific Parameters

Physical Layer LTSSM-specific parameters are listed in Table 6-8.

Table 6-8 LTSSM-specific parameters

Parameter Name	Settable VAR?*	Type	Range	Default	Description
SKIP_INITIAL_LINK_TRAINING					
	Yes	Integer	0-1	0	Allows link training to be completely skipped. The VIP will enter L0 once lane alignment has been achieved.
SKIP_POLLING_ACTIVE					
	Yes	Integer	0-1	0	Allows Polling.Active to be skipped during link training. This helps speed up link training for cases not specifically testing the ltssm states.
NUM_RX_FTS_REQUIRED_2_5G					
	Yes	Integer	0-255	255	The minimum number of FTS ordered sets required for a receiver to infer lock when exiting from electrical idle at 2.5GT/s.
NUM_RX_FTS_REQUIRED_5G					
	Yes	Integer	0-255	255	The minimum number of FTS ordered sets required for a receiver to infer lock when exiting from electrical idle at 5GT/s.
NUM_TX_FTS_REQUIRED_2_5G					
	Yes	Integer	0-255	255	When SKIP_INITIAL_LINK_TRAINING is set, this is the default N_FTS value which the LTSSM will use at 2.5G when exiting from P1. All subsequent exits will use a value obtained from training sets used during link training.

Table 6-8 LTSSM-specific parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
NUM_TX_FTS_REQUIRED_5G					
	Yes	Integer	0-255	255	When SKIP_INITIAL_LINK_TRAINING is set, this is the default N_FTS value which the LTSSM will use at 5G when exiting from P1. All subsequent exits will use a value obtained from training sets used during link training.
SET_TS_COMPLIANCE_RECEIVE					
	Yes	Integer	0-1	0	When set, outgoing training sets will have the compliance receive bit set.
TRANSMIT_MODIFIED_COMPLIANCE					
	Yes	Integer	0-1	0	When set, the VIP will transmit the modified compliance pattern upon entry into Polling.Compliance.
DETECT_QUIET_TIMEOUT_NS					
	Yes	Integer	200 - large value	120_000ns	Timeout interval for Detect.Quiet state.
DETECT_QUIET_SPEED_CHANGE_NS					
	Yes	Integer	200 - large value	1000ns	If the LTSSM enters Detect.Quiet at a speed other than 2.5G, this is the time the LTSSM will state in Detect.Quiet to complete a speed change back to 2.5G regardless of whether data is being received.
DETECT_ACTIVE_TIMEOUT_NS					
	Yes	Integer	200 - large value	120_000ns	Timeout interval for the Detect.Active state.
POLLING_ACTIVE_TIMEOUT_NS					
	Yes	Integer	200 - large value	240_000ns	If at least one but not all unconfigured lanes detect a receiver, this configures the time between receiver detect sequences.
POLLING_CONFIGURATION_TIMEOUT_NS					
	Yes	Integer	200 - large value	480_0000ns	Timeout limit for the Polling.Configuration state.
CONFIGURATION_LINKWIDTH_START_TIMEOUT_NS					
	Yes	Integer	200 - large value	240_000ns	Timeout value in the Configuration.Linkwidth state

Table 6-8 LTSSM-specific parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
CONFIGURATION_LINKWIDTH_ACCEPT_TIMEOUT_NS					
	Yes	Integer	200 - large value	20_000ns	Timeout value in the Configuration.Linkwidth state
CONFIGURATION_LANENUM_WAIT_TIMEOUT_NS					
	Yes	Integer	200 - large value	20_000ns	Timeout value in the Configuration.:Lanenum.Wait state
CONFIGURATION_COMPLETE_TIMEOUT_NS					
	Yes	Integer	200 - large value	20_000ns	Timeout value in the Configuration.:Configuraton.Complete state.
CONFIGURATION_IDLE_TIMEOUT_NS					
	Yes	Integer	200 - large value	20_000ns	Timeout value in the Configuration.Idle state
RECOVERY_RCVRLOCK_TIMEOUT_NS					
	Yes	Integer	200 - large value	240_000ns	Timeout in Recovery.RcvrLock state.
RECOVERY_RVCRCFG_TIMEOUT_NS					
	Yes	Integer	200 - large value	240_000ns	Timeout value in Recovery.RcvrCfg
RECOVERY_IDLE_TIMEOUT_NS					
	Yes	Integer	200 - large value	20_000ns	Timeout value in Recovery.Idle state.
HOT_RESET_TIMEOUT_NS					
	Yes	Integer	200 - large value	200_000ns	Timeout value in Hot_Reset state.
DISABLED_TIMEOUT_NS					
	Yes	Integer	200 - large value	2_000ns	Time before a downstream component will exit the Disabled state if no EIOS is detected
PHYSTATUS_TIMEOUT_NS					
	Yes	Integer	10 - large value	1_000ns	When VIP pl is a pipe master, this is the time the pl will wait for a PHYSTATUS to go high to acknowledge a receiver detect, a rate change, or powerdown change.

Table 6-8 LTSSM-specific parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
ENABLE_EXTENDED_SYNCH					
	Yes	Integer	0-1	0	When set, the transmitter for each lane must transmit 1024 TS1 ordered sets before transitioning from Recovery.RcvrLock to Recover.RcvrConfig.
MIN_NUM_TX_EIEOS_BEFORE_FTS					
	Yes	Integer	0-16	4	Minimum number of EIEOS ordered sets transmitted before transmitting FTS. (5GT/s only)
MAX_NUM_TX_EIEOS_BEFORE_FTS					
	Yes	Integer	0-16	4	Maximum number of EIEOS ordered sets transmitted before transmitting FTS. (5GT/s only)
NUM_TX_TS1_IN_POLLING_ACTIVE					
	Yes	Integer	16-4096	1024	Sets the number of training sets the LTSSM must transmit in Polling.Active before exiting this state.
RECOVERY_SPEED_ELECTRICAL_IDLE_TIME_IN_NS					
	Yes	Integer	100 - large value	1000	Number of ns the transmitter is required to be transmitting electrical idle during a speed change once the receiver has detected/inferred electrical idle in the state RECOVERY_SPEED.
INFERRED_ELECTRICAL_IDLE_SKP_TIME_NS					
	Yes	Integer	200 - large value	12800	The amount of time in L0 before the LTSSM will infer electrical idle if no SKP ordered sets are received.
ALLOW_RATE_POWER_SIMULTANEOUS_CHANGE					
	Yes	Integer	0-1	1	When set to a 1, the VIP in pipe slave mode will not flag rate and powerdown signals changing simultaneously. When set to 0, simultaneous rate and powerdown changes will be flagged.
ENABLE_RELAXED_SKP_CHECKING					
	Yes	Integer	0-1	0	When set, the VIP will not flag skp ordered sets received with greater or less than 3 skp symbols. This should only be set when there are repeaters or switches between the root and endpoint.
USER_TX_TS1_LANE_MASK					
	Yes	Integer	Any 32-bit value	32'hFFFF_FFFF	A bitwise lane enable for user training sets. A 'b1 enables user training sets for a particular lane and 'b0 disables user training sets. Bit 0 of this vector corresponds to lane 0, bit 1 corresponds to lane 1 and so on. Refer to the user training set tasks section for more details.

Table 6-8 LTSSM-specific parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
USER_TX_TS2_LANE_MASK					
	Yes	Integer	Any 32-bit value	32'hFFFF_FFFF	A bitwise lane enable for user training sets. A 'b1 enables user training sets for a particular lane and 'b0 disables user training sets. Bit 0 of this vector corresponds to lane 0, bit 1 corresponds to lane 1 and so on. Refer to the user training set tasks section for more details.
IS_TX_DOWNSTREAM					
	No	Integer	0-1	0	Indicates whether the transmitter is downstream or not. This affects link training behavior.
IS_PIPE_MASTER					
	No	Integer	0-1	1	When set to a 1, the VIP will behave as a pipe master. When set to 0, the VIP will behave as a pipe slave.
ENABLE_EQUALIZATION_PHASE					
	Yes	Integer	0. 1, 3	3	Enable the last equalization phase. NOTE: 0 will disable equalization.
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with "_VAR". This is useful if using higher level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.					

6.6.3 Equalization Parameters

Equalization parameters are listed in Table 6-9.

Table 6-9 Equalization parameters

Parameter Name	Settable VAR?	Default	Description
ENABLE_EQUALIZATION_VERIFICATION_MODE	YES	0	Enable equalization checking. See other Vars/Interface tasks to setup proper checking in SVC.
ENABLE_EQUALIZATION_PHASE	YES	3	Enable the last equalization phase. Valid values = 0, 2, 3. 0 will disable equalization.
NUM_ADDITIONAL_TS_EQ_BEFORE_TRANS_EQ0	YES	0	Number of additional TS1 to transmit before transitioning out of phase 0 equalization.
NUM_ADDITIONAL_TS_EQ_BEFORE_TRANS_EQ0	YES	0	Number of additional TS1 to transmit before transitioning out of phase 1 equalization.

Table 6-9 Equalization parameters (Continued)

Parameter Name	Settable VAR?	Default	Description
NUM_ADDITIONAL_TS_EQ_BEFORE_TRANS_EQ0	YES	0	Number of additional TS1 to transmit before transitioning out of phase 2 equalization.
NUM_ADDITIONAL_TS_EQ_BEFORE_TRANS_EQ0	YES	0	Number of additional TS1 to transmit before transitioning out of phase 3 equalization.
RECOVERY_EQUALIZATION_PHASE0_UPSTREAM_ TIMEOUT_NS	YES	12,000	Timeout for upstream lanes in phase 0 of recovery.equalization. Default = Spec_value/1000
RECOVERY_EQUALIZATION_PHASE1_UPSTREAM_ TIMEOUT_NS	YES	12,000	Timeout for upstream lanes in phase 1 of recovery.equalization. Default = Spec_value/1000
RECOVERY_EQUALIZATION_PHASE2_UPSTREAM_ TIMEOUT_NS	YES	24,000	Timeout for upstream lanes in phase 2 of recovery.equalization. Default = Spec_value/1000
RECOVERY_EQUALIZATION_PHASE3_UPSTREAM_ TIMEOUT_NS	YES	32_000	Timeout for upstream lanes in phase 3 of recovery.equalization. Default = Spec_value/1000
RECOVERY_EQUALIZATION_PHASE1_ DOWNSTREAM_TIMEOUT_NS	YES	24,000	Timeout for upstream lanes in phase 1 of recovery.equalization. Default = Spec_value/1000
RECOVERY_EQUALIZATION_PHASE2_ DOWNSTREAM_TIMEOUT_NS	YES	32_000	Timeout for upstream lanes in phase 2 of recovery.equalization. Default = Spec_value/1000
RECOVERY_EQUALIZATION_PHASE3_ DOWNSTREAM_TIMEOUT_NS	YES	24,000	Timeout for upstream lanes in phase 3 of recovery.equalization. Default = Spec_value/1000
REJECT_ PRESET_COEFF_REQUEST	Yes	0	Each bit corresponds to a lane. Valid only in Equalization Phase 2 for Downstream Port and Phase 3 for Upstream port. Setting a bit forces the corresponsding lane for side of the link receiving preset/coefficient requests to reject all new requests.
SET_TX_TS1_RESET_EIEOS_INTERVAL_COUNT_ BIT	Yes	0	When set and transmitting at 8G in equalization phases, TS1s are sent with the Reset_EIEOS_Interval_Count bit set.

Table 6-9 Equalization parameters (Continued)

Parameter Name	Settable VAR?	Default	Description
FULL_SWING	NO	6’d48	Default fs value port will advertise in outgoing EQTS. It is recommended to use the SetTxTS1FSLF task to change this value rather than using a defparam.
LOW_FREQUENCY_COEFFICIENT	NO	‘d3	Default LF value port will advertise in outgoing EQTS. It is recommended to use the SetTxTS1FSLF task to change this value rather than using a defparam.
PRECURSOR_COEFFICIENT	NO	0	Default precursor coefficient value that is loaded into all preset settings. Users should use the task SetEQPreset2CoeffTable to change preset to coefficient mappings.
CURSOR_COEFFICIENT	NO	36	Default cursor coefficient value that is loaded into all preset settings. Users should use the task SetEQPreset2CoeffTable to change preset to coefficient mappings.
POSTCURSOR_COEFFICIENT	NO	12	Default postcursor coefficient value that is loaded into all preset settings. Users should use the task SetEQPreset2CoeffTable to change preset to coefficient mappings.

6.7 Physical Layer Events

Pysical Layer events are listed in Table 6-10.

Table 6-10 Physical Layer events

Event Name	Description	Usage
event_rx_new_preset_coefficients	Event triggered when 2 TS1 OS are received with requested presets and/or coefficients. Triggered in Phase 2 for SVC as Downstream port and Phase 3 for SVC as Upstream port.	Can be used to determine when to call GetEQRequestedPresetCoefficients.
event_eq_phase_complete	An equalization phase has completed. Triggered each time phase 0, 1, 2, or 3 completes.	Can be used to check the equalization_complete_var
event_eq_complete	Equalization has completed. Asserted on LTSSM transition from equalization state to Recovery.RcvrLock.	

6.8 Physical Layer ASCII Signals

Physical Layer ASCII signals are listed in Table 6-11.

Table 6-11 Physical Layer ASCII signals

Signal Name	Description
ascii_ltssm_tx_state	LTSSM state of the transmitter
ascii_ltssm_rx_state	LTSSM state of the receiver
ascii_lane*n*_rx_data	Data received on lane n, where n is a number between 0 and 31.
ascii_lane*n*_tx_data	Data transmitted on lane n, where n is a number between 0 and 31.
ascii_pipe_lane*n*_rx_data;	Symbol data on received on PIPE lane *n*, where n is a number between 0 and 31.
ascii_pipe_lane*n*_tx_data	Symbol data on transmitted on PIPE lane *n*, where n is a number between 0 and 31.
ascii_hotplug_mode	Current state of hotplug mode
ascii_prev_hotplug_mode	Previous state of the hotplug.
ascii_lane_reversal_mode;	Lane reversal indicates if lane reversal is enabled.

6.9 Physical Layer Statistics

Types of Physical Layer statistics are listed in Table 6-12.

Table 6-12 Physical Layer Statistics

Stat Name	Width	Description
stat_num_rx_skp_ordered_sets	32	Number of skip ordered sets received on configured lanes
stat_num_tx_skp_ordered_sets	32	Number of skip ordered sets transmitted on configured lanes.
stat_num_tx_ts1	Array of integers	Number of TS1 sent per lane.
stat_num_tx_ts2	Array of integers	Number of TS2 sent per lane.
stat_num_rx_ts1	Array of integers	Number of TS1 received per lane
stat_num_rx_ts2	Array of integers	Number of TS2 ordered sets received per lane
stat_num_rx_eios	Array of integers	Number of EIOS received per lane.
stat_num_rx_eieos	Array of integers	Number of EIEOS received per lane.
stat_num_rx_fts	Array of integers	Number of FTS received per lane.
stat_num_rx_compliance_pattern	Array of integers	Number of compliance pattern sequences received per lane.
stat_num_rx_modified_compliance_pattern	Array of integers	Number of modified compliance pattern sequences received per lane.
stat_num_loopback_pattern	Array of integers	Number of times the VIP detected its own loopback pattern while a loopback master.
stat_num_hot_resets_received	32	Number of times the LTSSM entered hot reset by receiving training sets with the hot reset bit set.
stat_num_hot_resets_initiated	32	Number of times the LTSSM initiated a hot reset.
stat_num_link_disabled	32	Number of times the link entered the DISABLED state.
stat_num_l0s_tx_entry	32	Number of times the transmit side of the link entered L0s.
stat_num_l0s_rx_entry	32	Number of times the receive side of the link entered L0s.
stat_num_l1_entry	32	Number of times the LTSSM entered the L1 low power state.
stat_num_link_up	32	Number of times the LTSSM declares the link up.
stat_num_link_down	32	Number of times the LTSSM changes the link from up to down.
stat_num_retrain_request	32	Number of times the LTSSM is requested to retrain the link.
stat_num_ei_tx_ts1_missing_comma	Array of integers	Number of times a TS1 missing comma EI was injected on each lane.
stat_num_ei_tx_ts1_corrupt_link_number	Array of integers	Number of times a TS1 link number EI was injected on each lane.
stat_num_ei_tx_ts1_corrupt_lane_number	Array of integers	Number of times a TS1 lane number EI was injected on each lane.

Table 6-12 Physical Layer Statistics (Continued)

Stat Name	Width	Description
stat_num_ei_tx_ts1_corrupt_n_fts	Array of integers	Number of times a TS1 N_FTS EI was injected on each lane.
stat_num_ei_tx_ts1_corrupt_data_ rate	Array of integers	Number of times a TS1 data rate EI was injected on each lane
stat_num_ei_tx_ts1_corrupt_ training_control	Array of integers	Number of times a TS1 training control EI was injected on each lane
stat_num_ei_tx_ts1_corrupt_ identifier	Array of integers	Number of times a TS1 identifier EI was injected on each lane
stat_num_ei_tx_ts1_make_ts2	Array of integers	Number of times a make TS2 EI was injected on each lane.
stat_num_ei_tx_ts1_corrupt_ disparity	Array of integers	Number of times a TS1 corrupt disparity EI was injected on each lane
stat_num_ei_tx_ts1_invalid_ codeword	Array of integers	Number of times a TS1 invalid codeword EI was injected on each lane
stat_num_ei_tx_ts2_missing_ comma	Array of integers	Number of times a TS2 missing comma EI was injected on each lane.
stat_num_ei_tx_ts2_corrupt_link_ number	Array of integers	Number of times a TS2 link number EI was injected on each lane.
stat_num_ei_tx_ts2_corrupt_lane_number	Array of integers	Number of times a TS2 lane number EI was injected on each lane.
stat_num_ei_tx_ts2_corrupt_n_fts	Array of integers	Number of times a TS2 N_FTS EI was injected on each lane.
stat_num_ei_tx_ts2_corrupt_data_ rate	Array of integers	Number of times a TS2 data rate EI was injected on each lane
stat_num_ei_tx_ts2_corrupt_ training_control	Array of integers	Number of times a TS2 training control EI was injected on each lane
stat_num_ei_tx_ts2_corrupt_ identifier	Array of integers	Number of times a TS2 identifier EI was injected on each lane
stat_num_ei_tx_ts2_make_ts1	Array of integers	Number of times a make TS1 EI was injected on each lane.
stat_num_ei_tx_ts2_corrupt_ disparity	Array of integers	Number of times a TS2 corrupt disparity EI was injected on each lane
stat_num_ei_tx_ts2_invalid_ codeword	Array of integers	Number of times a TS2 invalid codeword EI was injected on each lane
stat_num_eq_phase_[1|2|3]_successfu	32	Number of successful completions for equalization phase [1|2|3].
stat_num_eq_completions	32	Number of times Equalization has completed.
stat_num_rx_eq_preset_requests	array of integers	Number of preset requests received. Preset value changes and use_preset field is set.

Table 6-12 Physical Layer Statistics (Continued)

Stat Name	Width	Description
stat_num_rx_eq_coefficient_requests	array of integers	Number of coefficient requests received. At least 1 coefficient field changes and use_preset is deasserted.
stat_num_tx_eq_preset_requests	array of integers	Number of preset requests transmitted.
stat_num_tx_eq_coefficient_requests	array of integers	Number of coefficient requests transmitted.
stat_num_rx_eq_preset_requests_rejected	array of integers	Number of preset requests which were rejected.
stat_num_rx_eq_coefficient_requests_ rejected	array of integers	Number of coefficient requests which were rejected.
stat_num_eq_phase_1_successful	32	Number of successful completions for equalization phase [1|2|3].
stat_num_eq_phase_2_successful	32	Number of successful completions for equalization phase [1|2|3].
stat_num_eq_phase_3_successful	32	Number of successful completions for equalization phase [1|2|3].
stat_num_eq_completions	32	Number of times Equalization has completed.

6.10 LTSSM Error Injection Controls

Physical Layer LTSSM error injection control parameters are listed in Table 6-13.

Table 6-13 LTSSM error injection control parameters

Parameter Name	Type	Range	Description	DUT Action
EI_TX_TRAINING_PERCENTAGE				
	Integer	0-100	Percentage probability of a training type EI occurring	
EI_TX_TRAINING_FORCE_POLLING_ACTIVE_TIMEOUT_WEIGHT				
	Integer	Typically 0-1000	Force the LTSSM to stay in Polling-.Active state until it has timed out.	Dut should transition to either Detect.Quiet transmit electrical idle.
EI_TX_TRAINING_FORCE_POLLING_CONFIGURATION_TIMEOUT_WEIGHT				
	Integer	Typically 0-1000	Force the LTSSM to stay in Polling.Configuration until it has timed out.	DUT should transition to Detect after the timeout and cease transmitting.
EI_TX_TRAINING_FORCE_CONFIGURATION_LINKWIDTH_START_TIMEOUT_WEIGHT				
	Integer	Typically 0-1000	Force the LTSSM stay in Configuration.Linkwidth.Start until the timeout period has expired.	DUT should timeout and transition to Detect after its timer has expired. No TS1 or TS2 ordered sets should be detected after the timeout.
EI_TX_TRAINING_FORCE_CONFIGURATION_LINKWIDTH_ACCEPT_TIMEOUT_WEIGHT				

Table 6-13 LTSSM error injection control parameters (Continued)

Parameter Name	Type	Range	Description	DUT Action
	Integer	Typically 0-1000	Force the LTSSM to stay in Linkwidth.Accept until the timeout period has expired.	Dut should timeout and transition to Detect after its timer has expired. No TS1 or TS2 ordered sets should be sent after transitioning to Detect.
EI_TX_TRAINING_FORCE_CONFIGURATION_LANENUM_ACCEPT_TIMEOUT_WEIGHT				
	Integer	Typically 0-1000	Force the LTSSM to stay in Configuration.Lanenum.Accept until the timeout period has expired.	Dut should timeout and transition to detect after its timer has expired.
EI_TX_TRAINING_CONFIGURATION_COMPLETE_TIMEOUT_WEIGHT				
	Integer	Typically 0-1000	Force the LTSSM to stay in Configuration.Complete until the timeout period has expired.	Dut should timeout and transition to detect after the timer has expired. No TS1 or TS2 ordered sets should be sent after the timeout.
EI_TX_TRAINING_CONFIGURATION_IDLE_TIMEOUT_WEIGHT				
	Integer	Typically 0-1000	Force the LTSSM to stay in Configuration.Idle until the timeout period has expired.	Dut should timeout and transition to detect after a 2ms timer has expired. No transmission of any kind should be sent after the timeout.
EI_TX_TRAINING_FORCE_RECOVERY_RCVRLOCK_TIMEOUT_WEIGHT				
	Integer	Typically 0-1000	Force the LTSSM to stay in Recovery.Rcvrlock until the timeout period has expired.	Depending on conditions, the DUT should transition to Recovery.Rcvrcfg or Recovery.Speed.
EI_TX_TRAINING_FORCE_RECOVERY_RCVRCFG_TIMEOUT_WEIGHT				
	Integer	Typically 0-1000	Force the LTSSM to stay in Recoveyr.Rvcrcfg until the timeout period has expired.	Dut should transition to the detect state
EI_TX_TRAINING_FORCE_RECOVERY_IDLE_TIMEOUT_WEIGHT				
	Integer	Typically 0-1000	Force the LTSSM to stay in the Recovery.Idle until the timeout period has expired.	Dut may initially transition to L0, but after LTSSM has timed out the DUT should detect ordered sets and transition to Recovery.Rcvrlock when it detects training ordered sets.

6.11 TS1 Error Injection Controls

6.11.1 TS1 Randomized Injections

Controls that can be used to randomize by weighting various types of errors are listed in Table 6-14.

Table 6-14 TS1 error injection control parameters

Parameter Name	Type	Range	Description	DUT Action
EI_TX_TS1_PERCENTAGE				
	Integer	0-100	Percentage probability of a TS1 type EI occurring	
EI_TX_TS1_LANE_MASK				
	Integer	Any 32-bit value	A bitwise enable mask for only applying TX_TS1 EIs to certain lanes. Default value is 32'hFFFF_FFFF (all lanes enabled).	
EI_TX_TS1_MISING_COMMA_WEIGHT				
	Integer	Typically 0-1000	Changes the starting K28.5 of a TS1 into random data.	DUT should not detect a TS1 and reset its consecutive training set count.
EI_TX_TS1_CORRUPT_LINK_NUMBER_WEIGHT				
	Integer	Typically 0-1000	If PAD is supposed to be sent, the assigned link number will be sent instead. If the link number is supposed to be sent, then PAD will be sent instead.	This EI should reset the consecutive count of training sets and halt link training progress.
EI_TX_TS1_CORRUPT_LANE_NUMBER_WEIGHT				
	Integer	Typically 0-1000	If PAD is supposed to be sent, the assigned lane number will be sent instead. If the lane number is supposed to be sent, then PAD will be sent instead.	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS1_CORRUPT_N_FTS_WEIGHT				
	Integer	Typically 0-1000	Sends a PAD in place of data on the N_FTS byte of a training set.	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS1_CORRUPT_DATA_RATE_WEIGHT				
	Integer	Typically 0-1000	Sends a PAD in place of data on the data rate field of a training set.	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS1_CORRUPT_TRAINING_CONTROL_WEIGHT				
	Integer	Typically 0-1000	Sends PAD in place of the training control field of a training set.	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS1_CORRUPT_IDENTIFIER_WEIGHT				
	Integer	Typically 0-1000	Sends random data in place of one of the training control identifiers (bytes 6-15 of a training set).	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS1_MAKE_TS2_WEIGHT				

Table 6-14 TS1 error injection control parameters (Continued)

Parameter Name	Type	Range	Description	DUT Action
	Integer	Typically 0-1000	Changes all of the TS1 identifier byte of a TS1 set (bytes 6-15) into TS2 identifier bytes.	DUT should detect a TS2 instead of a TS1. The net result of detecting a TS2 ordered set is state specific.
EI_TX_TS1_CORRUPT_DISPARITY_WEIGHT				
	Integer	Typically 0-1000	Sends the wrong disparity on a random data byte of a training set.	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS1_INVALID_CODEWORD_WEIGHT				
	Integer	Typically 0-1000	Replaces random data byte of a training set with an invalid codeword.	This EI should reset the consecutive count of received training sets and halt link training progress.

6.11.2 Directed Sequences of TS1 Injected Errors

For advanced LTSSM testing, the `SetEiTxTs1Pattern` directed task can be used to send specified sequences of good and bad training sets. The `SetEiTxTs1Pattern` task is described in Table 6-15.

In order to use this task, EI_TX_TS1_PERCENTAGE_VAR must be set to 0. All other weights and the lane mask need to be set in order to choose what type of error will be selected on bad TS1s

Table 6-15 Task to send sequences of training sets

Task Name	Arguments (length)	I/O	Values
SetEiTxTs1Pattern	enable_i (32)	I	‘1’ enables the training set pattern, ‘0’ disables it
Defines a constrained random burst of TS1 ordered sets with errors.	min_ei_tx_ts1_burst_i	I	Minimum number of bad TS1 sets in the error burst.
	max_ei_tx_ts1_burst_i	I	Maximum number of bad TS1 sets in the error burst.
	min_ei_tx_ts1_spacing_i	I	Minimum number of good TS1 sets between the error burst.
	max_ei_tx_ts1_spacing_i	I	Minimum number of good TS1 sets between the error burst.

6.12 TS2 Error Injection Controls

6.12.1 TS2 Randomized Injections

Controls that can be used to randomize by weighting various types of errors are listed in Table 6-16.

Table 6-16 TS2 randomized injection control parameters

Parameter Name	Type	Range	Description	DUT Action
EI_TX_TS2_PERCENTAGE				
	Integer	0-100	Percentage probability of a TS2 type EI occurring	
EI_TX_TS2_LANE_MASK				
	Integer	Any 32-bit value	A bitwise enable mask for only applying TX_TS1 EIs to certain lanes. Default value is 32'hFFFF_FFFF (all lanes enabled).	
EI_TX_TS2_MISSING_COMMA_WEIGHT				
	Integer	Typically 0-1000	Changes the starting K28.5 of a TS1 into random data.	DUT should not detect a TS1 and reset its consecutive training set count.
EI_TX_TS2_CORRUPT_LINK_NUMBER_WEIGHT				
	Integer	Typically 0-1000	If PAD is supposed to be sent, the assigned link number will be sent instead. If the link number is supposed to be sent, then PAD will be sent instead.	This EI should reset the consecutive count of training sets and halt link training progress.
EI_TX_TS2_CORRUPT_LANE_NUMBER_WEIGHT				
	Integer	Typically 0-1000	If PAD is supposed to be sent, the assigned lane number will be sent instead. If the lane number is supposed to be sent, then PAD will be sent instead.	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS2_CORRUPT_N_FTS_WEIGHT				
	Integer	Typically 0-1000	Sends a PAD in place of data on the N_FTS byte of a training set.	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS2_CORRUPT_DATA_RATE_WEIGHT				
	Integer	Typically 0-1000	Sends a PAD in place of data on the data rate field of a training set.	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS2_CORRUPT_TRAINING_CONTROL_WEIGHT				
	Integer	Typically 0-1000	Sends PAD in place of the training control field of a training set.	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS2_CORRUPT_IDENTIFIER_WEIGHT				

Table 6-16 TS2 randomized injection control parameters (Continued)

Parameter Name	Type	Range	Description	DUT Action
	Integer	Typically 0-1000	Sends random data in place of one of the training control identifiers (bytes 6-15 of a training set).	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS2_MAKE_TS1_WEIGHT				
	Integer	Typically 0-1000	Changes all of the TS2 identifier byte of a TS1 set (bytes 6-15) into TS21 identifier bytes.	DUT should detect a TS1 instead of a TS2. The net result of detecting a TS1 ordered set is state specific.
EI_TX_TS2_CORRUPT_DISPARITY_WEIGHT				
	Integer	Typically 0-1000	Sends the wrong disparity on a random data byte of a training set.	This EI should reset the consecutive count of received training sets and halt link training progress.
EI_TX_TS2_INVALID_CODEWORD_WEIGHT				
	Integer	Typically 0-1000	Replaces random data byte of a training set with an invalid codeword.	This EI should reset the consecutive count of received training sets and halt link training progress.

6.12.2 Directed Sequences of TS2 Injected Errors

For advanced LTSSM testing, the `SetEiTxTs2Pattern` directed task (see Table 6-17) can be used to send specified sequences of good and bad training sets. In order to use this task, EI_TX_TS2_PERCENTAGE_VAR must be set to 0. All other weights and the lane mask need to be set in order to choose what type of error will be selected on bad TS1s.

Table 6-17 SetEiTxTs2Pattern task

Task Name	Arguments (length)	I/O	Values
SetEiTxTs2Pattern	enable_i (32)	I	‘1’ enables the training set pattern, ‘0’ disables it
Defines a constrained random burst TS1 ordered sets with errors.	min_ei_tx_ts1_burst_i	I	Minimum number of bad TS2 sets in the error burst.
	max_ei_tx_ts1_burst_i	I	Maximum number of bad TS2 sets in the error burst.
	min_ei_tx_ts1_spacing_i	I	Minimum number of good TS2 sets between the error burst.
	max_ei_tx_ts1_spacing_i	I	Minimum number of good TS2 sets between the error burst.

6.13 Ordered Set Error Injection Controls

6.13.1 EIOS Injected Errors

Controls that can be used to inject errors into EIOS blocks are listed in Table 6-18.

Table 6-18 EIOS injected errors

Parameter Name	Settable VAR*	Default Value	Description
EI_TX_EIOS_PERCENTAGE	YES	0	Percentage probability of an EIOS type EI occurring
EI_TX_EIOS_LANE_MASK	YES	32’hFFFF_FFFF	A bitwise enable mask for only applying TX_EIOS EIs to certain lanes.
EI_TX_EIOS_INVALID_DATA_BYTE0_WEIGHT	YES	0	Changes the first byte of an EIOS to randomly generated data.
EI_TX_EIOS_INVALID_DATA_BYTE1_WEIGHT	YES	0	Changes byte one of the EIOS to randomly generated data.
EI_TX_EIOS_INVALID_DATA_BYTE2_WEIGHT	YES	0	Changes byte two of the EIOS to randomly generated data.
EI_TX_EIOS_INVALID_DATA_BYTE3_WEIGHT	YES	0	Changes byte three of the EIOS to randomly generated data.
EI_TX_EIOS_INVALID_DATA_BYTE4_WEIGHT	YES	0	Changes byte four of the EIOS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_EIOS_INVALID_DATA_BYTE5_WEIGHT	YES	0	Changes byte five of the EIOS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_EIOS_INVALID_DATA_BYTE6_WEIGHT	YES	0	Changes byte six of the EIOS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_EIOS_INVALID_DATA_BYTE7_WEIGHT	YES	0	Changes byte seven of the EIOS symbol to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_EIOS_INVALID_DATA_BYTE8_WEIGHT	YES	0	Changes byte eight of the EIOS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_EIOS_INVALID_DATA_BYTE9_WEIGHT	YES	0	Changes byte nine of the EIOS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_EIOS_INVALID_DATA_BYTE10_WEIGHT	YES	0	Changes byte ten of the EIOS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_EIOS_INVALID_DATA_BYTE11_WEIGHT	YES	0	Changes byte eleven of the EIOS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_EIOS_INVALID_DATA_BYTE12_WEIGHT	YES	0	Changes byte twelve of the EIOS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_EIOS_INVALID_DATA_BYTE13_WEIGHT	YES	0	Changes byte thirteen of the EIOS to randomly generated data. Valid only at speeds of 8G and above.

Table 6-18 EIOS injected errors (Continued)

EI_TX_EIOS_INVALID_DATA_BYTE14_WEIGHT	YES	0	Changes byte fourteen of the EIOS set to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_EIOS_INVALID_DATA_BYTE15_WEIGHT	YES	0	Changes byte 15 of the EIOS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_EIOS_CORRUPT_DISPARITY_BYTE0_WEIGHT	YES	0	Inserts a disparity error on byte 0 of the ordered set
EI_TX_EIOS_CORRUPT_DISPARITY_BYTE1_WEIGHT	YES	0	Inserts a disparity error on byte 1 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_EIOS_CORRUPT_DISPARITY_BYTE2_WEIGHT	YES	0	Inserts a disparity error on byte 2 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_EIOS_CORRUPT_DISPARITY_BYTE3_WEIGHT	YES	0	Inserts a disparity error on byte 3 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_EIOS_INVALID_CODEWORD_BYTE0_WEIGHT	YES	0	Inserts an invalid codeword on byte 0 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_EIOS_INVALID_CODEWORD_BYTE1_WEIGHT	YES	0	Inserts an invalid codeword on byte 1 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_EIOS_INVALID_CODEWORD_BYTE2_WEIGHT	YES	0	Inserts an invalid codeword on byte 2 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_EIOS_INVALID_CODEWORD_BYTE3_WEIGHT	YES	0	Inserts an invalid codeword on byte 3 of the ordered set. Valid at 2.5G and 5G speeds only.

6.13.2 EEIOS Injected Errors

Controls that can be used to inject errors into EEIOS blocks are listed in Table 6-19.

Table 6-19 EEIOS injected errors

Parameter Name	Settable VAR*	Default Value	Description
EI_TX_EIEOS_PERCENTAGE	YES	0	Percentage probability of an EIOS type EI occurring
EI_TX_EIEOS_LANE_MASK	YES	32’hFFFF_FFFF	A bitwise enable mask for only applying TX_EIEOS EIs to certain lanes.
EI_TX_EIEOS_INVALID_DATA_BYTE0_WEIGHT	YES	0	Changes byte 0 of the EIEOS ordered set to randomly generated data.

Table 6-19 EEIOS injected errors (Continued)

EI_TX_EIEOS_INVALID_DATA_BYTE1_WEIGHT	YES	0	Changes byte one of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE2_WEIGHT	YES	0	Changes byte two of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE3_WEIGHT	YES	0	Changes byte three of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE4_WEIGHT	YES	0	Changes byte four of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE5_WEIGHT	YES	0	Changes byte five of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE6_WEIGHT	YES	0	Changes byte six of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE7_WEIGHT	YES	0	Changes byte seven of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE8_WEIGHT	YES	0	Changes byte eight of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE9_WEIGHT	YES	0	Changes byte nine of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE10_WEIGHT	YES	0	Changes byte ten of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE11_WEIGHT	YES	0	Changes byte eleven of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE12_WEIGHT	YES	0	Changes byte twelve of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE13_WEIGHT	YES	0	Changes byte thirteen of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE14_WEIGHT	YES	0	Changes byte fourteen of the EIEOS to randomly generated data.
EI_TX_EIEOS_INVALID_DATA_BYTE15_WEIGHT	YES	0	Changes byte 15 symbol to some other randomly generated data value
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE0_WEIGHT	YES	0	Inserts a disparity error on byte 0 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE1_WEIGHT	YES	0	Inserts a disparity error on byte 1 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE2_WEIGHT	YES	0	Inserts a disparity error on byte 2 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE3_WEIGHT	YES	0	Inserts a disparity error on byte 3 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE4_WEIGHT	YES	0	Inserts a disparity error on byte 4 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE5_WEIGHT	YES	0	Inserts a disparity error on byte 5 of the ordered set

Table 6-19 EEIOS injected errors (Continued)

EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE6_WEIGHT	YES	0	Inserts a disparity error on byte 6 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE7_WEIGHT	YES	0	Inserts a disparity error on byte 7 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE8_WEIGHT	YES	0	Inserts a disparity error on byte 8 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE9_WEIGHT	YES	0	Inserts a disparity error on byte 9 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE10_WEIGHT	YES	0	Inserts a disparity error on byte 10 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE11_WEIGHT	YES	0	Inserts a disparity error on byte 11 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE12_WEIGHT	YES	0	Inserts a disparity error on byte 12 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE13_WEIGHT	YES	0	Inserts a disparity error on byte 13 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE14_WEIGHT	YES	0	Inserts a disparity error on byte 14 of the ordered set
EI_TX_EIEOS_CORRUPT_DISPARITY_BYTE15_WEIGHT	YES	0	Inserts a disparity error on byte 15 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE0_WEIGHT	YES	0	Inserts an invalid codeword on byte 0 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE1_WEIGHT	YES	0	Inserts an invalid codeword on byte 1 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE2_WEIGHT	YES	0	Inserts an invalid codeword on byte 2 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE3_WEIGHT	YES	0	Inserts a disparity error on byte 3 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE4_WEIGHT	YES	0	Inserts an invalid codeword on byte 4 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE5_WEIGHT	YES	0	Inserts an invalid codeword on byte 5 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE6_WEIGHT	YES	0	Inserts an invalid codeword on byte 6 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE7_WEIGHT	YES	0	Inserts an invalid codeword on byte 7 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE8_WEIGHT	YES	0	Inserts an invalid codeword on byte 8 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE9_WEIGHT	YES	0	Inserts an invalid codeword on byte 9 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE10_WEIGHT	YES	0	Inserts an invalid codeword on byte 10 of the ordered set

Table 6-19 EEIOS injected errors (Continued)

EI_TX_EIEOS_INVALID_CODEWORD_BYTE11_WEIGHT	YES	0	Inserts an invalid codeword on byte 11 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE12_WEIGHT	YES	0	Inserts an invalid codeword on byte 12 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE13_WEIGHT	YES	0	Inserts an invalid codeword on byte 13 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE14_WEIGHT	YES	0	Inserts an invalid codeword on byte 14 of the ordered set
EI_TX_EIEOS_INVALID_CODEWORD_BYTE15_WEIGHT	YES	0	Inserts an invalid codeword on byte 15 of the ordered set

6.13.3 FTS Injected Errors

FTS are sent exclusively in the state Tx.L0s.Exit. FTS injected errors are listed in Table 6-20.

Table 6-20 FTS injected errors

Parameter Name	Settable VAR*	Default Value	Description
EI_TX_FTS_PERCENTAGE	YES		Percentage probability of an FTS type EI occurring
EI_TX_FTS_LANE_MASK	YES	32'hFFFF _FFFF	A bitwise enable mask for only applying TX_FTS EIs to certain lanes.
EI_TX_FTS_INVALID_DATA_BYTE0_WEIGHT	YES	0	Changes the first byte of an FTS to randomly generated data.
EI_TX_FTS_INVALID_DATA_BYTE1_WEIGHT	YES	0	Changes byte one of the FTS to randomly generated data.
EI_TX_FTS_INVALID_DATA_BYTE2_WEIGHT	YES	0	Changes byte two of the FTS to randomly generated data.
EI_TX_FTS_INVALID_DATA_BYTE3_WEIGHT	YES	0	Changes byte three of the FTS to randomly generated data.
EI_TX_FTS_INVALID_DATA_BYTE4_WEIGHT	YES	0	Changes byte four of the FTS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_FTS_INVALID_DATA_BYTE5_WEIGHT	YES	0	Changes byte five of the FTS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_FTS_INVALID_DATA_BYTE6_WEIGHT	YES	0	Changes byte six of the FTS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_FTS_INVALID_DATA_BYTE7_WEIGHT	YES	0	Changes byte seven of the FTS symbol to randomly generated data. Valid only at speeds of 8G and above.

Table 6-20 FTS injected errors (Continued)

EI_TX_FTS_INVALID_DATA_BYTE8_WEIGHT	YES	0	Changes byte eight of the FTS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_FTS_INVALID_DATA_BYTE9_WEIGHT	YES	0	Changes byte nine of the FTS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_FTS_INVALID_DATA_BYTE10_WEIGHT	YES	0	Changes byte ten of the FTS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_FTS_INVALID_DATA_BYTE11_WEIGHT	YES	0	Changes byte eleven of the FTS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_FTS_INVALID_DATA_BYTE12_WEIGHT	YES	0	Changes byte twelve of the FTS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_FTS_INVALID_DATA_BYTE13_WEIGHT	YES	0	Changes byte thirteen of the FTS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_FTS_INVALID_DATA_BYTE14_WEIGHT	YES	0	Changes byte fourteen of the FTS set to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_FTS_INVALID_DATA_BYTE15_WEIGHT	YES	0	Changes byte 15 of the FTS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_FTS_CORRUPT_DISPARITY_BYTE0_WEIGHT	YES	0	Inserts a disparity error on byte 0 of the ordered set
EI_TX_FTS_CORRUPT_DISPARITY_BYTE1_WEIGHT	YES	0	Inserts a disparity error on byte 1 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_FTS_CORRUPT_DISPARITY_BYTE2_WEIGHT	YES	0	Inserts a disparity error on byte 2 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_FTS_CORRUPT_DISPARITY_BYTE3_WEIGHT	YES	0	Inserts a disparity error on byte 3 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_FTS_INVALID_CODEWORD_BYTE0_WEIGHT	YES	0	Inserts an invalid codeword on byte 0 of the ordered set. Valid at 2.5G and 5G speeds only.

Table 6-20 FTS injected errors (Continued)

EI_TX_FTS_INVALID_CODEWORD_BYTE1_WEIGHT	YES	0	Inserts an invalid codeword on byte 1 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_FTS_INVALID_CODEWORD_BYTE2_WEIGHT	YES	0	Inserts an invalid codeword on byte 2 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_FTS_INVALID_CODEWORD_BYTE3_WEIGHT	YES	0	Inserts an invalid codeword on byte 3 of the ordered set. Valid at 2.5G and 5G speeds only.

6.13.4 SKP Injected Errors

The error injections and controls listed in Table 6-21 assume that the user is transmitting SKPs with the lengths specified in the PCIE standard (4 bytes for 8/b10b encoding and 16bytes for 128/130b encoding). There are controls to make SKPs longer or shorter than the normal length, but it is assumed for the purposes of injecting errors that standard lengths are used.

Table 6-21 SKP injected errors

Parameter Name	Settable VAR*	Default Value	Description
EI_TX_SKP_PERCENTAGE	YES	0	Percentage probability of an SKP type EI occurring
EI_TX_SKP_LANE_MASK	YES	32’hFFFF _FFFF	A bitwise enable mask for only applying TX_SKP EIs to certain lanes.
EI_TX_SKP_INVALID_DATA_BYTE0_WEIGHT	YES	0	Changes the first byte of an SKP to randomly generated data.
EI_TX_SKP_INVALID_DATA_BYTE1_WEIGHT	YES	0	Changes byte one of the SKP to randomly generated data.
EI_TX_SKP_INVALID_DATA_BYTE2_WEIGHT	YES	0	Changes byte two of the SKP to randomly generated data.
EI_TX_SKP_INVALID_DATA_BYTE3_WEIGHT	YES	0	Changes byte three of the SKP to randomly generated data.
EI_TX_SKP_INVALID_DATA_BYTE4_WEIGHT	YES	0	Changes byte four of the SKP to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SKP_INVALID_DATA_BYTE5_WEIGHT	YES	0	Changes byte five of the SKP to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SKP_INVALID_DATA_BYTE6_WEIGHT	YES	0	Changes byte six of the SKP to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SKP_INVALID_DATA_BYTE7_WEIGHT	YES	0	Changes byte seven of the SKP symbol to randomly generated data. Valid only at speeds of 8G and above.

Table 6-21 SKP injected errors (Continued)

EI_TX_SKP_INVALID_DATA_BYTE8_WEIGHT	YES	0	Changes byte eight of the SKP to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SKP_INVALID_DATA_BYTE9_WEIGHT	YES	0	Changes byte nine of the SKP to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SKP_INVALID_DATA_BYTE10_WEIGHT	YES	0	Changes byte ten of the SKP to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SKP_INVALID_DATA_BYTE11_WEIGHT	YES	0	Changes byte eleven of the EIOS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SKP_INVALID_DATA_BYTE12_WEIGHT	YES	0	Changes byte twelve of the SKP to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SKP_INVALID_DATA_BYTE13_WEIGHT	YES	0	Changes byte thirteen of the SKP to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SKP_INVALID_DATA_BYTE14_WEIGHT	YES	0	Changes byte fourteen of the SKP set to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SKP_INVALID_DATA_BYTE15_WEIGHT	YES	0	Changes byte 15 of the SKP to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SKP_CORRUPT_DISPARITY_BYTE0_WEIGHT	YES	0	Inserts a disparity error on byte 0 of the ordered set
EI_TX_SKP_CORRUPT_DISPARITY_BYTE1_WEIGHT	YES	0	Inserts a disparity error on byte 1 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_SKP_CORRUPT_DISPARITY_BYTE2_WEIGHT	YES	0	Inserts a disparity error on byte 2 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_SKP_CORRUPT_DISPARITY_BYTE3_WEIGHT	YES	0	Inserts a disparity error on byte 3 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_SKP_INVALID_CODEWORD_BYTE0_WEIGHT	YES	0	Inserts an invalid codeword on byte 0 of the ordered set. Valid at 2.5G and 5G speeds only.

Table 6-21 SKP injected errors (Continued)

EI_TX_SKP_INVALID_CODEWORD_BYTE1_WEIGHT	YES	0	Inserts an invalid codeword on byte 1 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_SKP_INVALID_CODEWORD_BYTE2_WEIGHT	YES	0	Inserts an invalid codeword on byte 2 of the ordered set. Valid at 2.5G and 5G speeds only.
EI_TX_SKP_INVALID_CODEWORD_BYTE3_WEIGHT	YES	0	Inserts an invalid codeword on byte 3 of the ordered set. Valid at 2.5G and 5G speeds only.

6.13.5 SDS Injected Errors

The errors listed in Table 6-22 are defined for the start of data stream ordered set.

Table 6-22 SDS injected errors

Parameter Name	Settable VAR*	Default Value	Description
EI_TX_SDS_PERCENTAGE	YES	0	Percentage probability of an SDS type EI occurring
EI_TX_SDS_LANE_MASK	YES	32'hFFFF _FFFF	A bitwise enable mask for only applying TX_SDS EIs to certain lanes.
EI_TX_SDS_INVALID_DATA_BYTE0_WEIGHT	YES	0	Changes the first byte of an SDS to randomly generated data.
EI_TX_SDS_INVALID_DATA_BYTE1_WEIGHT	YES	0	Changes byte one of the SDS to randomly generated data.
EI_TX_SDS_INVALID_DATA_BYTE2_WEIGHT	YES	0	Changes byte two of the SDS to randomly generated data.
EI_TX_SDS_INVALID_DATA_BYTE3_WEIGHT	YES	0	Changes byte three of the SDS to randomly generated data.
EI_TX_SDS_INVALID_DATA_BYTE4_WEIGHT	YES	0	Changes byte four of the SDS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SDS_INVALID_DATA_BYTE5_WEIGHT	YES	0	Changes byte five of the SDS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SDS_INVALID_DATA_BYTE6_WEIGHT	YES	0	Changes byte six of the SDS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SDS_INVALID_DATA_BYTE7_WEIGHT	YES	0	Changes byte seven of the SDS symbol to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SDS_INVALID_DATA_BYTE8_WEIGHT	YES	0	Changes byte eight of the SDS to randomly generated data. Valid only at speeds of 8G and above.

Table 6-22 SDS injected errors (Continued)

EI_TX_SDS_INVALID_DATA_BYTE9_WEIGHT	YES	0	Changes byte nine of the SDS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SDS_INVALID_DATA_BYTE10_WEIGHT	YES	0	Changes byte ten of the SDS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SDS_INVALID_DATA_BYTE11_WEIGHT	YES	0	Changes byte eleven of the SDS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SDS_INVALID_DATA_BYTE12_WEIGHT	YES	0	Changes byte twelve of the SDS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SDS_INVALID_DATA_BYTE13_WEIGHT	YES	0	Changes byte thirteen of the SDS to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SDS_INVALID_DATA_BYTE14_WEIGHT	YES	0	Changes byte fourteen of the SDS set to randomly generated data. Valid only at speeds of 8G and above.
EI_TX_SDS_INVALID_DATA_BYTE15_WEIGHT	YES	0	Changes byte 15 of the SDS to randomly generated data. Valid only at speeds of 8G and above.

6.13.6 EDS Injected Errors

The EDS token replaces the last 4 bytes of the data stream preceding an ordered set with the bytes {‘h1f, 8’h80, 8’h90, 8’h00}. This is set on the last 4 bytes of the block, and thus the location of these symbols will vary based on link width. EDS injected errors are listed in Table 6-23.

Table 6-23 EDS injected errors

Parameter Name	Settable VAR*	Default Value	Description
EI_TX_EDS_PERCENTAGE	YES	0	Percentage probability of an FTS type EI occurring
EI_TX_EDS_INVALID_DATA_BYTE0_WEIGHT	YES	0	Changes the first byte of an EDS to randomly generated data.
EI_TX_EDS_INVALID_DATA_BYTE1_WEIGHT	YES	0	Changes byte one of the EDS to randomly generated data.
EI_TX_EDS_INVALID_DATA_BYTE2_WEIGHT	YES	0	Changes byte two of the EDS to randomly generated data.
EI_TX_EDS_INVALID_DATA_BYTE3_WEIGHT	YES	0	Changes byte three of the EDS to randomly generated data.

6.13.7 IDLE Data Injected Errors

IDLE data error injections are not injected randomly, so there is an enable control rather than a percentage. This EI is meant to be used in directed test cases where the user specifically needs to prevent logical idle or IDL tokens from being received. These EIs work on logical idle symbols at 2.5/5G and on IDL tokens at 8G. IDLE data injected errros are listed in Table 6-24.

Table 6-24 IDLE data injected errors

Parameter Name	Settable VAR*	Default Value	Description
EI_TX_IDLE_DATA_ENABLE	YES	0	Enables the injection of errors onto IDLE data or IDL tokens.
EI_TX_IDLE_DATA_LANE_MASK	YES	32'hFFFF _FFFF	Lane mask vector for IDLE_DATA EI's. A bit set to 0 will prevent any errors being injected on the corresponding lane
EI_TX_IDLE_DATA_INVALID_BYTE_WEIGHT	YES	0	Changes the byte of an IDLE DATA to randomly generated non-zero data. Valid at all speeds.
EI_TX_IDLE_DATA_CORRUPT_DISPARITY _WEIGHT	YES	0	Inserts a disparity error on byte of IDLE DATA. Valid at 2.5G and 5G speeds only.
EI_TX_IDLE_DATA_INVALID_CODEWORD_WEIGHT	YES	0	Inserts an invalid codeword on byte IDLE DATA. Valid at 2.5G and 5G speeds only.

6.13.8 Directed Sequences of IDLE Injected Errors

For advanced LTSSM testing, the SetEiTxIdleDataPattern directed task can be used to send specified sequences of good and bad idle symbols. In order to use this task, EI_TX_IDLE_DATA_ENABLE_VAR must be set to 0. All other weights and the lane mask need to be set in order to choose what type of error will be selected on bad IDLE symbols. Arguments to the SetEiTxIdleDataPattern task are listed in Table 6-25.

Table 6-25 SetEiTxIdleDataPattern task

Task Name	Arguments	I/O	Values
SetEiTxIdleDataPattern	enable_i(32)	I	'1' enables the training set pattern, '0' disables it
Defines a constrained random burst of IDLE symbols with errors.	min_ei_tx_ts1_burst_i	I	Minimum number of bad IDLE symbols in the error burst.
	max_ei_tx_ts1_burst_i	I	Maximum number of bad idle symbols in the error burst.
	min_ei_tx_ts1_spacing_i	I	Minimum number of good idle symbols between the error burst.
	max_ei_tx_ts1_spacing_i	I	Minimum number of good idle symbols between the error burst.

6.14 Physical Layer Internal Interface Tasks

Physical Layer internal interface tasks are listed in Table 6-26.

Table 6-26 Physical Layer internal interface tasks

Task Name	Arguments (length)	I/O	Values
TransmitPhy	data_in (256)	I	One byte of outgoing data per lane.
	control_bus (256)	I	Indication of whether the outgoing byte is valid or not, or whether it is a start of TLP, end of TLP, etc. Each byte of the control bus corresponds to a byte of data.
	ei_code (256)	I	Error injection code. There is a 1 byte error injection code per byte of data.
	hold (1)	O	Indication to the link that the phy cannot accept data.
	force_align_packet (1)	O	Indication to the link that it should align the next start of packet to byte 0 of the next symbol time.
ReceivePhy	data_out (256)	O	One byte of outgoing data per lane.
	control_bus (256)	O	Indication of whether the outgoing byte is valid or not, or whether it is a start of TLP, end of TLP, etc. Each byte of the control bus corresponds to a byte of data.
	ei_code (256)	O	Error injection code. There is a 1 byte error injection code per byte of data.
TransmitPCS	pcs_lanenum_i (32)	I	Lane which the transmit data goes on
	txd_o (32)	O	Transmit data. If all 4 bytes are utilized, txd[7:0] contains the byte which is to be transmitted first, txd[15:8] is to be transmitted second, and so on.
	txc_o (4)	O	Control bit. txc[0] corresponds to txd[7:0] and so on.
	ei_code_o (32)	O	Error injection code
	tx_elec_idle_o (1)	O	Transmit electrical idle PIPE signal.
	tx_compliance_o (1)	O	Instructs the PCS to send the compliance pattern to the serdes.
	tx_data_valid_o (1)	O	Indication that data transmitted by the VIP is valid. Used primarily for rate matching. NOTE: pipe 3 only.
	tx_start_block_o (1)	O	Bitwise indication that a new ordered set/data block will start on this byte. NOTE: pipe 3 only.
	tx_sync_header_o (2)	O	The sync header to be inserted when a new block is started. Only used at 8GT/s.
	invert_tx_polarity_o (1)	O	Indication to the PCS to invert polarity on the transmit side for testing purposes.
	rx_status_o (3)	O	When VIP is in the pipe slave configuration, this is the status that accompanies the outgoing data. Not valid in other configurations.

Table 6-26 Physical Layer internal interface tasks (Continued)

Task Name	Arguments (length)	I/O	Values
ReceivePCS	pcs_lanenum (32)	I	PCS number which the data came from.
	rxd_i (32)	I	Sent data. If all four bytes are used, rxd[7:0] was sent first and rxd[31:24] was sent most recently.
	rxc_i (32)	I	Control bit for the receive data. Bit 0 corresponds to the low byte of data and bit 3 corresponds to the high byte.
	ei_code_i (32)	I	Receive error injection code
	rx_status_i (3)	I	Receiver status encoding: 3'b000 – Receive OK 3'b001 – 1 SKP added 3'b010 – 1 SKP removed 3'b011 – Receiver detected 3'b100 – 128/130B decode error and/or disparity error. 3'b101 – Elastic buffer overflow 3'b110 – Elastic buffer underflow 3'b111 – Receive disparity error
	rx_valid_i (1)	I	Indication that the receiver has detected valid data and achieved symbol lock.
	rx_data_valid_i (1)	I	Indication that data coming into the VIP is valid. Used primarily for rate matching. NOTE: pipe 3 only.
	rx_elec_idle_i (1)	I	Indication that electrical idle is being received into the VIP Note: Any used ports must be tied to a 1.
	rx_start_block_i (1)	I	Indication that data coming into the VIP is the first byte of data in the block. NOTE: pipe 3 only.
	rx_elec_idle_i (1)	I	Indication that the receiver has detected electrical idle.
	rx_polarity_i (1)	I	Pipe slave signal. Determined if DUT has detected polarity inversion. Not valid in pipe master config.
	rx_polarity_o (1)	O	Receive polarity inversion. Directs receive PCS decoder to invert polarity on incoming symbols. Used only as a pipe master, not valid when VIP is configured as a pipe slave. 1'b0 – No polarity inversion should be performed 1'b1 - Polarity inversion should occur

7 Physical Coding Sublayer (2.5Gb / 5Gb/ 8Gb)

The Physical Coding Sublayer (PCS) checks for commas and performs symbol lock at 2.5GT/s, 5GT/s, and 8GT/s (Optional). It also does 8b/10b encoding at 2.5GT/s and 5GT/s and inserts data sync headers at 8GT/s. The PCS also contains the elastic buffer which inserts/deletes SKP symbols from skip ordered sets. The PCS task pair interface and module inputs/outputs comply with Intel's PIPE interface specification. The PCS module-level inputs/outputs are listed in Table 7-1.

Table 7-1 PCS module-level I/Os

Name	I/O	Description
reset	I(1)	Active high reset. Must only be asserted for 100ns ONCE per simulation at the beginning.
data_bus_width	I(2)	Indicates the width of the pipe data bus.
rate	I(2)	Pipe rate signal.
powerdown	I(2)	Power state of the transmitter/receiver pair.
detect_receiver	I(1)	Indication from the phy to perform a receiver detect.
loopback_enable	I(1)	Indication from the Physical Layer to loop back incoming data on the transmitter.
block_align_control	I(1)	Controls slave block alignment. NOTE: pipe 3 only.
receiver_present	I(1)	Indication to the PCS that it should indicate a receiver is present.
serdes_locked	I(1)	Indication from the attached serdes that the receiver has locked onto the incoming bit stream.

7.1 Physical Coding Sublayer Parameters

PCS parameters are listed in Table 7-2.

Table 7-2 PCS parameters

Parameter Name	Settable VAR?*	Type	Range	Default	Description
COMMA_SYNC_COUNT					
	Yes	Integer	1-16	3	Number of commas that must be detected in the same bit position to declare successful symbol lock.
ENABLE_RX_ELASTIC_BUFFER					
	No	Integer	0-1	1	Enables the receive elastic buffer. This is not necessary for most simulations.
NUM_PMA_INTERFACE_BITS					
	No	Integer	10, 16, 32, 64, 128	10	Number of bits on the PMA interface. NOTE: please set this in the model.
DISPLAY_NAME					
	No	String		pciesvc_pcs	Default display name for msglog statements.
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with "_VAR". This is useful if using higher level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.					

7.2 Physical Coding Sublayer Interface Tasks

PCS interface tasks are listed in Table 7-3.

Table 7-3 PCS interface tasks

Task Name	Arguments (length)	I/O	Values
TransmitPCS	txd_i (32)	I	Transmit data. If the 32-bit interface is used, byte txd[7:0] is the byte to be transmitted first, txd[15:8] is to be transmitted second, and so on.
	txc_i (4)	I	Control bit. Each bit corresponds to a byte in txd.
	ei_code (32)	I	Error injection code
	tx_elec_idle_i (1)	I	Indication for transmitter to send electrical idle.
	tx_compliance_i (1)	I	Indication to transmit the compliance pattern.
	tx_data_valid_i (1)	I	Indication that data transmitted by the VIP is valid. Used primarily for rate matching. NOTE: pipe 3 only.
	tx_start_block_ i(1)	I	Bitwise indication that a new ordered set/data block will start on this byte. NOTE: pipe 3 only.
	tx_sync_header_i (2)	I	The sync header to be inserted when a new block is started. NOTE: pipe 3 only.
	invert_tx_polarity_i (1)	I	Indication to the encoder to invert transmit polarity.

Table 7-3 PCS interface tasks (Continued)

Task Name	Arguments (length)	I/O	Values
ReceivePCS	rxd_o (32)	O	One byte of outgoing data per lane.
	rxc_o (4)	O	Control bit for the receive data. Bit 0 corresponds to the low byte of data and bit 3 corresponds to the high byte.
	ei_code_o (32)	O	Error injection code
	rx_status_o (3)	O	Receive status: 3'b000: RX data OK 3'b001: 1 SKP added to elastic buffer 3'b010: 1 skp removed from elastic buffer 3'b011: Receiver detected 3'b100: 8b/10b decode error and rx disparity error. 3;b101: elastic buffer overflow 3'b110: elastic buffer underflow 3'b111: rx disparity error
	rx_valid_o (1)	O	Indication that the receiver has detected valid data and achieved symbol lock.
	rx_data_valid_o (1)	O	Indication that data received by the VIP is valid. Used primarily for rate matching. NOTE: pipe 3 only.
	rx_start_block_o (1)	O	Bitwise indication that a new ordered set/data block will start on this byte. NOTE: pipe 3 only.
	rx_sync_header_o (2)	O	The sync header to be inserted when a new block is started. NOTE: pipe 3 only.
	rx_polarity_i (1)	I	Polarity inversion. 0 – Phy does not perform polarity inversion. 1 – Phy performs polarity inversion.
TransmitPMA	data_o(NUM_PMA_ INTERFACE_BITS)	O	Data to transmit to serdes
	tx_elec_idle_o (1)	O	Indication for serdes to transmit electrical idle.
ReceivePMA	data_i(NUM_PMA_ INTERFACE_BITS)	I	Incoming 8b/10b encoded (2.5GT/s & 5GT/s) or scrambled data (8GT/s)

8 Serializer/Deserializer (pciesvc_serdes.v)

The Serializer/Deserializer (SERDES) is used to convert the serial, differential bit stream into a stream of 10-bit bytes. In addition, it also does signal-level validation, receiver clock-recovery and bit alignment.

The SERDES model supplied is not an analog SERDES model, but a digital representation, and is not meant for verifying an analog SERDES design. It is provided so that Phy Layer functionality such as speed negotiation may be tested.

8.1 Module Port List

The SERDES module is instantiated by the models. It may also be used by a DUT to alleviate the simulation performance penalty often observed from timing information laden analog models. The ports are listed in Table 8-1.

Table 8-1 SERDES module ports

Port Name	I/O	Description
reset	I	Causes the SERDES to reset, positive active
tx_enable (1)	I	Indicates input bits are valid – also used to modulate the bit stream for OOB.
tx_bit_clk (1)	I	clock for transmitting tx_byte10
tx_byte10 (10)	I	Outbound transmit byte
rx_datap (1)	I	Bit stream of RX data
rx_datan (1)	I	Bit stream of RX data (differential).
pll_locked (1)	O	PLL/DLL has locked to incoming bit stream
serdes_locked (1)	O	SERDES has locked to incoming COMMA characters
rbc (2)	O	Receive bit clock
rx_byte10_clk (1)	O	Receive byte clock
rx_byte10 (10)	O	Sent byte
comma_det (1)	O	Indicates COMMA character detected

Table 8-1 SERDES module ports (Continued)

Port Name	I/O	Description
tx_datap (1)	O	Bit stream of TX data
tx_datan (1)	O	Bit stream of TX data (differential).
sig_level_valid (1)	O	Signal level deemed valid – also used to receive bit stream for OOB

8.2 SERDES Parameters

There are a few parameters that affect the behavior of the SERDES. These parameters are listed in Table 8-2.

Table 8-2 SERDES parameters

Parameter Name	Settable VAR?*	Type	Range	Default	Description
COMMA_SYNC_COUNT					
	No	Integer	0 - large value	0	Number of aligned commas before SERDES lock declared. Not used in PCIE.
BIT_SYNC_COUNT					
	No	Integer	10 - large value	1000	Number of min bit periods seen before PLL lock declared. Must be larger than maximum time of OOB active burst.
CLK_TOLERANCE					
	No	Integer	Any value	0.0001	Tolerance in ns. Equivalent to 100 PPM SAS requirement. This should not be changed as it affects SSC clock tracking.
USE_ZERO_FOR_DISABLED					
	Yes	Integer	0-1	0 (FALSE)	If set, the SERDES when disabled will transmit n = p = 0 (ground).
USE_Z_FOR_DISABLED					
	Yes	Integer	0-1	1 (TRUE)	If set, the SERDES when disabled will transmit n = p = Z (high impedance). Note: If neither USE_ZERO_FOR_DISABLED nor USE_Z_FOR_DISABLED are set, then n = p = random (no spread).
COMMA_P					
	No	Integer	Any 10-bit value	10’b0011 111010	Definition of positive COMMA character.
COMMA_N					
	No	Integer	Any 10-bit value	10’b1100 000101	Definition of negative COMMA character.

Table 8-2 SERDES parameters (Continued)

Parameter Name	Settable VAR?*	Type	Range	Default	Description
DISPLAY_NAME					
	No	String		pciesvc_serdes.	String prefixed to messages to display in the output log.
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with “_VAR”. This is useful if using higher level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.					

9 PCIE Verification Tools

9.1 The Global PCIE Address Space Shadow

The purpose of this component is to provide a shadow of your DUT's PCI Express address space (memory and configuration space) for Write and Read transaction checking. The Global PCIE Address Space Shadow is an optional component which you can instantiate in your test environment.

The VIP captures TLPs as they go on the wire and then are passed to the shadow's transaction handler which decodes all relevant transactions and updates the expected global shadow state. For example, outbound MemWrite TLPs are inspected and (if appropriate) written to the *global shadow memory.* This shadow memory can be used by any checkers to allow comparisons with actual completion data (from the CplD TLP) and the expected completion data (from the global shadow memory.)

If instantiated and enabled, the Global Shadow is used by both the driver and the requester to automatically verify the results of the memory and configuration accesses made to the DUT. Memory transactions are captured and recorded in the Memory Shadow; no initial configuration of the Global System Shadow is required in most cases.

The shadow also provides for "ignored" regions of the memory. Any ignored regions will return an attribute to this effect when an application queries the shadow for expected read results. This way a checker can determine if the transaction is expected to return a predicable result or not. Occasionally there will be memory regions (e.g. registers) that you do not want to check for correctness – write-only registers, status or statistics registers that may change sporadically, etc. These regions in the shadow memory can be marked as *IGNORED* via the *AddMemRange()* task.

The memory shadow has two ordering modes, the normal *non-ordered* mode, as well as a mode for strict transaction ordering. Strict ordering is only for use with DUTs that will **not** reorder any inbound transactions. The default mode is to not assume any ordering (which is normal per the PCI Express rules). These regions in the shadow memory can be marked as *ORDERED* via the `AddMemRange()` task.

Typically the system shadow is instantiated (usually at the top-level file, analogously to the `svc_mem`) as `global_shadow0`; with a path define set in the testcase to point to it. When multiple root hierarchies are in use (see above discussion on multiple root hierarchies), the hard-coded definition is not used and a lookup task will instead provide the handle to the appropriate hierarchy's shadow.

If the shadow memory is not instantiated, please make sure the following shadow-checking parameters/ _VARs are not enabled:

ENABLE_SHADOW_CFG_LOOKUP (Transaction layer. See Table 4-4)

ENABLE_TX_TLP_REPORTING (Driver application. See Table 3-1)

ENABLE_TX_TLP_REPORTING (Requester application. See Table 3-3)

ENABLE_TX_TLP_REPORTING (Data Link layer. See Table 3-1)

ENABLE_SHADOW_MEMORY_CHECKING (model configuration. See Table 1-4)

ENABLE_SHADOW_MEMORY_CHECKING (Driver application. Table 3-1)

ENABLE_SHADOW_MEMORY_CHECKING (Requester application. See Table 3-3)

9.1.1 Instantiating and Enabling the Global Shadow

In your top file instantiate the shadow memory (or memories) with:

```
`ifdef EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH    // single root hierarchy:
   pciesvc_global_shadow global_shadow0();
   defparam global_shadow0.DISPLAY_NAME = "global_shadow0.";

`else    // Multiple root hierarchies:
   // root0
   pciesvc_global_shadow global_shadow0();
   defparam global_shadow0.DISPLAY_NAME = "global_shadow0.";
   defparam global_shadow0.HIERARCHY_NUMBER=0;// match your root hierarchy
   // root1
   pciesvc_global_shadow global_shadow1();
   defparam global_shadow1.DISPLAY_NAME = "global_shadow1.";
   defparam global_shadow1.HIERARCHY_NUMBER=1;
`endif  // EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH
```

9.1.1.1 Accessing the Shadow Tasks – Single Root Hierarchy

In the case of a single root hierarchy, knowing where you instantiated the global shadow memory, you can then set the global define to provide access to this; either on the command line or in an .fl (filelist) file:

```
+define+EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH=top.global_shadow0
```

This provides the path to allow the various VIP components to access the global shadow memory. For example:

```
`EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH.GetExpectedReadData(
   mem_start_addr,
   dword_num,
   byte_enables,
   transaction_id,
   exp_word_data,
   status
);
```

See below for additional examples. See “Shadow Tasks” on page 179 for descriptions of the shadow tasks.

9.1.1.2 Accessing the Shadow Tasks – Multiple Root Hierarchies

In the case of multiple root hierarchies, make sure that the EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH define is not defined, but turn on the SystemVerilog API with the PCIESVC_INCLUDE_SYSTEMVERILOG_API define.   In your code, you then need to import to provide access to the global_shadow_api, then declare the handle to the global_shadow:

```
`ifdef PCIESVC_INCLUDE_SYSTEMVERILOG_API
   import pciesvc_api_pkg::pciesvc_global_shadow_api;
   pciesvc_global_shadow_api     global_shadow;// Handle
`endif  // PCIESVC_INCLUDE_SYSTEMVERILOG_API
```

Then to get the fill in the handle to the associated SystemVerilog Global Shadow API tasks, call the *LookupGlobalShadow()* task:

With this handle, you will then be able to access the various Shadow API tasks (recall that SystemVerilog API tasks have “Api” appended to their name, so an analogous example to the above would be:

```
pciesvc_global_shadow_api::LookupGlobalShadow(HIERARCHY_NUMBER_VAR,
   global_shadow);
if (global_shadow == null)
   begin
      $msglog(LOG_NOTE, "%sCheckReceivedData: Could not find a global shadow
      (hier_num=%0d)", DISPLAY_NAME, HIERARCHY_NUMBER_VAR);
   end
else
   begin
      global_shadow.GetExpectedReadDataApi(
      mem_start_addr,
      dword_num,
      byte_enables,
      transaction_id,
      exp_word_data,
      status);
end
```

See “Shadow Tasks” on page 179 for descriptions of the shadow tasks.

9.1.1.3 Modifying the Shadow Memory Configuration

Within the shadow memory a memory target component is instantiated. To modify the behavior of the shadow’s underlying memory, you will need to explicitly scope into that memory target at instantiation time. (See “Memory Target Parameters” for a detailed description of the target memory parameters.) Note that you typically will be setting parameters here. Modifying the shadow’s target memory could cause the shadow to become corrupted.

Here is an example of how to modify the number of 32-bit pages for the shadow memory:

```
    defparam global_shadow0.shadow_mem0.NUM_32_BIT_PAGES = 8192;
```

9.1.2 Shadow Tasks

Shadow tasks are accessed via the EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH definition discussed in “Instantiating and Enabling the Global Shadow”, and are used to configure and control the shadow memory. The shadow tasks are described in Table 9-1.

If you are using the SystemVerilog API version of the tasks (that is, if you have multiple root hierarchies), remember that these have “Api” appended to their name.

Table 9-1 Shadow tasks

Task Name	Arguments (length)	I/O	Values
AddMemRange Define a special region of memory in the Transaction Handler's shadow memory. Note that is only required if non-default attributes are required on the memory range. Currently the only attribute is "ignored".	min_range (64)	I	Minimum memory address for the region.
	max_range (64)	I	Maximum memory address for the region.
	attributes (32)	I	The attribute(s) for the specified region. Attributes are listed in the *pciesvc_application_parms.v* file.
	error (1)	O	Returned status for the AddMemRange.
RemoveMemRange Remove a special region of memory that was added previously. Note that the min/ max_range must exactly match that which was specified in AddMemRange.	min_range(64)	I	Minimum memory address for the region.
	max_range (64)	I	Maximum memory address for the region.
	attributes (32)	I	The attribute(s) for the specified region to be removed.
	error (1)	O	Returned error status for the RemoveMemRange call.
GetExpectedReadData Fetch the expected dword data for the given transaction. Typically called once per dword being checked, where the offset is in the range [0-data_length_in_dwords) and the byte_enables will match those of the request.	pcie_start_addr (64)	I	Base address for reading the expected data.
	offset (32)	I	Offset to the *pcie_start_addr* above. Valid offsets are from 0 to (*data_length_in_dwords-1*)
	byte_enables (4)	I	Data word byte enables.
	transaction_id (32)	I	The combination of *RequesterID* and *Tag* to keep track of a given transaction.
	data (32)	O	Dword of expected data being retrieved from the shadow.
	status (32)	O	Returned status for this call
TransactionComplete Tell the shadow that your application is finished with the identified transaction, and that it should commit or release any resources held for that transaction. This task must be called for every memory transaction that results in a returned completion (e.g. MemRd).	pcie_start_addr (64)	I	Base address for reading the expected data.
	length_in_dwords (32)	I	Dword length of the requested transaction (for the full transaction, not just one completion.
	transaction_id	I	The combination of *RequesterID* and *Tag* for the transaction to cleanup.
	status (32)	O	Returned status for this call

Table 9-1 Shadow tasks (Continued)

Task Name	Arguments (length)	I/O	Values
ReadShadowCfgReg Read the given configuration register for a particular device in the shadow configuration space.	bdf (16)	I	Bus-Device-Function for the device you are reading.
	cfg_reg (10)	I	The specific configuration-register number.
	data (32)	O	Value of the shadowed copy of the configuration register.
	status (32)	O	Returned status of the task (parameters are in the file *pciesvc_cfg_params.v*)
ReadShadowCfgCapReg Read the given configuration register for a particular device in the shadow configuration space.	bdf (16)	I	Bus-Device-Function for the device you are reading.
	cfg_cap (8)	I	The specific configuration-capability we are reading.
	cfg_reg (10)	I	The specific configuration-register offset.
	data (32)	O	Value of the shadowed copy of the configuration register.
	status (32)	O	Returned status of the task (parameters are in the file *pciesvc_cfg_params.v*)
ModifyShadowCfgReg Modify the given configuration register with the data provided. Note that the only bits that will be modified are those set in the data_mask.	bdf (16)	I	Bus-Device-Function for the device you are reading.
	cfg_reg (10)	I	The specific configuration-register number.
	data (32)	I	Data to be written to the configuration-register (qualified with the below *mask*.)
	data_mask (32)	I	Asserted bits indicate which bits of above *data* argument will be modified in the configuration-register (remaining bits will be unchanged.)
	status (32)	O	Returned status of the task (parameters are in the file *pciesvc_cfg_params.v*)
ModifyShadowCfgCapReg Modify the given configuration capabilities register with the data provided. Note that the only bits that will be modified are those set in the data_mask.	bdf (16)	I	Bus-Device-Function for the device you are reading.
	cfg_cap (8)	I	The specific configuration-capability we are modifying. **NOTE: the capability must already exist in the device's configuration database.**
	cfg_reg (10)	I	The specific configuration-register offset.
	data (32)	I	Data to be written to the configuration-register (qualified with the below *mask*.)
	data_mask (32)	I	Asserted bits indicate which bits of above *data* argument will be modified in the configuration-register (remaining bits will be unchanged.)
	status (32)	O	Returned status of the task (parameters are in the file *pciesvc_cfg_params.v*)

Note that each instantiation of the VIP Link Layer has its own means to enable the reporting of transmitted *TLP*s via the parameter/_VAR **ENABLE_TX_TLP_REPORTING**. This is set (enabled) by default.

The driver and requester applications also have the same **ENABLE_TX_TLP_REPORTING** parameter/ _VAR, however this should, in general, only be enabled when the corresponding VIP LinkLayer parameter is *not* used (e.g. when the driver/requester are driving a DUT, there isn't any instantiation of the VIP LinkLayer, so in order to capture the transmitted TLPs, this parameter/_VAR should be set in the driver and/or requester instantiations.)

9.1.3 Example Global Shadow Memory Usage

To use the global shadow memory, a few items need to be instantiated and configured, once this is done, the driver and requester will automatically check the memory transactions for data validity.

Set the handle to the tasks, either via the shadow define (EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH in "Instantiating and Enabling the Global Shadow") or the SystemVerilog handle (in "Accessing the Shadow Tasks – Multiple Root Hierarchies").

Setup the optional ignored memory ranges: by default, the memory in the shadow memory is allocated automatically upon writes to it. If, however, you need to define memory range that is "ignored" when checked (where the reads do not necessarily match the preceding writes – clear-on-write registers are one example.)

```
reg [63:0]  MIN_IGNORED_MEM_ADDR,
            MAX_IGNORED_MEM_ADDR;   //must be 64 bits wide
reg [31:0]   ignored_mem_attr;
reg          mem_error;   // returned error
ignored_mem_attr = 0;
// Various MEM_CHECK… params are in pciesvc_application_params.v
ignored_mem_attr[MEM_CHECK_ATTR_IGNORE_BIT];
// Ignore memory between 0x1_0000 for 64KB
MIN_IGNORED_MEM_ADDR = 64'h0000_0000_0001_0000;
MAX_IGNORED_MEM_ADDR = 64'h0000_0000_0001_ffff;
// Now add the ignored memory regions to the shadow memory. Note
// that we are only demonstrating the single-root method:
`EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH.AddMemRange(
   MIN_IGNORED_MEM_ADDR,
   MAX_IGNORED_MEM_ADDR,
   ignored_mem_attr, mem_error
);
// When the test-case, driver or requester compares the memory
// between the MIN/MAX_IGNORED_MEM_ADDR addresses, the shadow
// will return status of "ignored" to indicate that this
// comparison can be ignored.
```

Run your simulation. The VIP driver and requester will automatically check the results of read completions against the original written value (except, of course, the above ignored ranges.)

9.1.4 Example Global Shadow Configuration Usage

To use the global shadow configuration space, one must make sure the DUT's default configuration values are written, once this is done, the Transaction Layer will automatically check the transactions against the configuration shadow for validity.

Set the shadow-define (or the multiple-hierarchy setup) as described above.

Make sure any configuration registers in the Shadow Configuration Space match those in the DUT’s configuration space by one of the following methods:

- Writing to the DUTs configuration registers to initialize or change their behavior (which automatically will update the shadow copy).
- Using the *ModifyShadowCfgReg* and/or *ModifyShadowCfgCapReg* tasks to update the shadow configuration space of non-default startup values for your device – for example, the default value of the ENABLE_NO_SNOOP bit in the Device Control Register is ‘1’, if the DUT’s default is ‘0’, we need to tell the Global Shadow about that non-default behavior:

```
`ifdef EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH
  // Set the data value
  cfg_data = 0;  // Start with a fresh variable.
  cfg_data[CFG_CAP_DEV_CONTROL_REG_ENABLE_NO_SNOOP_BIT] = 0;
  // Set value mask off the bit(s) we want to modify
  cfg_data_mask = 0;
  cfg_data_mask[CFG_CAP_DEV_CONTROL_REG_ENABLE_NO_SNOOP_BIT]=1;
     //Set mask
   // Modify the bit(s)
   `EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH.ModifyShadowCfgCapReg(
      endpoint0_driver0_req_id,
      CFG0_PCIE_CAP_ID,
      CFG0_PCIE_CAP_DEV_CONTROL_REG,
      cfg_data,
      cfg_data_mask,
      cfg_status
   );
`endif  //  EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH
```

Run your simulation. The VIP will check values of the configuration for appropriate behavior.

Note: Currently only a subset of the configuration registers is initialized.

9.2 The Transaction Logger

All inbound and outbound transactions to/from the VIP (both TLPs and DLLPs) are sent to the *transaction logger*. These transactions are distilled and written (one-transaction-per-line) to the transaction log file.

By default, the transaction logger is disabled. To enable it, and cause it to start writing transactions to a file, call the *$msglog_control* task:

```
$msglog_control(
   MSGLOG_CONTROL_SET_TRANSACTION_LOG_FILE,
   0,   // unused argument
   "logfile "
);      // log file or “” to end logging
```

The “*logfile*” is either a relative or full-path name filename. If the log file is given as the empty string (“”), the transaction log is closed. See also the Msglog documentation (in Util/Doc) for additional information regarding $msglog_control and other related tasks.

9.3 The Protocol Analyzer Logger

The VIP supports the Synopsys Protocol Analyzer (PA) tool, which allows users to view transactions, TLPs, DLLPs, ordered sets and the LTSSM state machine graphically. A transaction is made up of one request TLP and if necessary one or more completion TLPs required to complete that trasaction.

PA logging is disabled by default. In order to enable PA logging, the variable PCIESVC_INCLUDE_PA must be defined, and the file Include/pciesvc_pa_pkg.sv must be added to the compile list. There are tasks located in the Transaction Layer, the Data Link Layer, and the Physical Layer to enable and disable which items are logged. All of the parameters that are used below are located in Include/pciesvc_parms.sv.

Table 9-2 describes the Protocol Analyzer tasks in the Transaction Layer.

To invoke Protocol Analyzer, do:

```
% $DESIGNWARE_HOME/bin/pa
```

Documentation is available at

$DESIGNWARE_HOME/vip/tools/pa/latest/doc/protocol_analyzer_getting_started_guide.pdf

For more information, see to the Protocol Analyzer documentation.

Table 9-2 Protocol Analyzer tasks in the Transaction Layer

Task Name	Arguments (length)	I/O	Values
SetPATransactionMode Enables or disables the logging of transactions for the PA tool. Transaction logging is disabled by default.	mode (32)	I	Configures logging for transactions. PA_TRANSACTION_MODE_TRANSACTION_LOGGING_ENABLED- Transaction logging is enabled. PA_TRANSACTION_MODE_TRANSACTION_LOGGING_ENABLED- Transaction logging is disabled.
SetPALogFile Sets the path and name of the output file. The default file name is *instance_name*.pcie_svc.xml, where *instance_name* is the hierarchical path of the device model.	string filename	I	Path and filename where the xml file should be logged. Example: SetPALogfile("output/my_file.pcie_svc.xml") will redirect the logfile to output/my_file.pcie_svc.xml. NOTE: If changing the default filename, it is recommended that the new logfile end in .pcie_svc_.xml so that the PA tool reads in the log properly.

Control of TLP/DLLP logging is located in the Data Link Layer. Table 9-3 describes the Protocol Analyzer tasks in the Data Link Layer.

Table 9-3 Protocol Analyzer tasks in the Data Link Layer

Task Name	Arguments (length)	I/O	Values
SetPALinkMode Enables or disables the logging of TLPs and DLLPs for the PA tool. TLP/DLLP logging is disabled by default.	mode (32)	I	Configures logging for TLP/DLLPs. PA_LINK_MODE_PACKET_LOGGING_DISABLED- TLP/DLLP logging is disabled. PA_LINK_MODE_PACKET_LOGGING_ENABLED- TLP/DLLP logging is enabled PA_LINK_MODE_PAYLOAD_LOGGING_ENABLED- Enabled the logging of payloads when TLP/DLLP logging is enabled.

The Physical Layer has a task to control the logging of ordered sets and the LTSSM states. Table 9-4 describes the Protocol Analyzer task in the Physical Layer.

Table 9-4 Protocol Analyzer tasks in the Physical Layer

Task Name	Arguments (length)	I/O	Values
SetPAPhyMode Enables or disables the logging of ordered sets and LTSSM states for the PA tool. Ordered set and LTSSM logging is disabled by default.	mode (32)	I	Configures logging for ordered sets and the LTSSM state machine. PA_PHY_MODE_LOGGING_DISABLED – all logging in the phy is disabled PA_PHY_MODE_OS_LOGGING_ENABLED- Logging of ordered sets is enabled PA_PHY_MODE_LTSSM_LOGGING_ENABLED- Logging of LTSSM states is enabled.

Only one configuration parameter can be enabled at a time, so if you want to enable multiple items from the tables above, you must OR the parameters together. Below is an example of how to enable logging for the PA tool:

```
top.root0.tl0.SetPATransactionMode(
   PA_TRANSACTION_MODE_TRANSACTION_LOGGING_ENABLED
);
// Enable logging of TLPs and DLLPs with payload:
top.root0.dl0.SetPALinkMode(
   PA_LINK_MODE_PACKET_LOGGING_ENABLED |
   PA_LINK_MODE_PAYLOAD_LOGGING_ENABLED
);
// Enable logging of ordered sets only in the phy layer:
top.root0.pl0.SetPAPhyMode(PA_PHY_MODE_OS_LOGGING_ENABLED);
```

Protocol Analyzer Verilog example

```
`ifdef PCIESVC_INCLUDE_PA
   // Enable logging for transactions
   top.root0.port0.tl0.SetPALogfile(“output/root.xml”);
   top.root0.port0.tl0.SetTransactionMode(
      PA_TRANSACTION_MODE_TRANSACTION_LOGGING_ENABLED
   );
   // Collect TLP/DLLP and payload data
   top.root0.port0.dl0.SetPALinkMode(
      PA_LINK_MODE_PACKET_LOGGING_ENABLED |
      PA_LINK_MODE_PAYLOAD_LOGGING_ENABLED
   );
   // Collect ordered sets and LTSSM information
   top.root0.port0.pl0.SetPAPhyMode(
      PA_PHY_MODE_OS_LOGGING_ENABLED |
      PA_PHY_MODE_LTSSM_LOGGING_ENABLED
   );
`endif
```

Protocol Analyzer UVM example

```
virtual task run_phase(uvm_phase phase) ;
   …
   pciesvc_phy_set_pa_phy_mode_item set_pa_phy_mode;
   pciesvc_dl_set_pa_link_mode_item set_pa_link_mode;
   pciesvc_tl_set_pa_transaction_mode_item
      set_pa_transaction_mode;
   pciesvc_tl_set_pa_logfile_item set_pa_logfile;

   // Create sequence items
   set_pa_phy_mode =
      pciesvc_phy_set_pa_phy_mode_item::type_id::create(
         "set_pa_phy_mode"
      );
   set_pa_link_mode =
      pciesvc_dl_set_pa_link_mode_item::type_id::create(
         "set_pa_link_mode"
      );
   set_pa_transaction_mode =
      pciesvc_tl_set_pa_transaction_mode_item::type_id::creat
(
         "set_pa_transaction_mode"
      );
   set_pa_logfile =
pciesvc_tl_set_pa_logfile_item::type_id::create(
      “set_pa_logfile”
   );
// Enable Protocol Analyzer options
   set_pa_phy_mode.mode = PA_PHY_MODE_OS_LOGGING_ENABLED |
     PA_PHY_MODE_LTSSM_LOGGING_ENABLED;
   set_pa_link_mode.mode=PA_LINK_MODE_PAYLOAD_LOGGING_ENABLED |
      PA_LINK_MODE_PACKET_LOGGING_ENABLED;
```

```
    set_pa_transaction_mode.mode =
       PA_TRANSACTION_MODE_TRANSACTION_LOGGING_ENABLED;
    set_pa_logfile.filename = “output/root.xml”;
    // Set values by executing
   env.root0.port[0].pl[0].control_agent.sequencer.execute_item   (
       set_pa_phy_mode
    );
   env.root0.port[0].dl[0].control_agent.sequencer.execute_item
    (
       set_pa_link_mode
    );
   env.root0.port[0].tl[0].control_agent.sequencer.execute_item
    (
       set_pa_transaction_mode
    );
   env.root0.port[0].tl[0].control_agent.sequencer.execute_item
    (
      set_pa_logfile
    );
endtask
```

9.4 Symbol Logging

All symbol traffic in and out of the VIP can be captured at the PIPE and recorded in the symbol log file. To enable symbol logging, set the phy layer parameter ENABLE_SYMBOL_LOG to 1. For example:

```
  defparam    top.endpoint0.port0.pl0.ENABLE_SYMBOL_LOG  = 1;
```

By default this parameter is 0 so symbol logging is disabled.

In simulations with multiple instances of the VIP, all symbol log information is recorded in the same file. Each line of symbol data includes the instance name, so it is possible to separate out the information for a specific instance using standard utilities like grep.

Symbol traffic is captured at the PIPE interface, even if the VIP interface to the simulation is not the PIPE. The PIPE datapath width may be configured as 1, 2 or 4 bytes. If the width is greater than 1 byte, all bytes that are captured on a given cycle are logged at the same time in the symbol log.

Symbols are logged from the perspective of the VIP: symbols received by the VIP are logged on the RX side and symbols transmitted by the vip are logged on the TX side.

The symbol log contains three additional types of information:

- It shows the current LTSSM state for the VIP. In states where the RX and TX state may differ it shows both states.
- If encoding, disparity, or bit flip errors are injected on the TX side, it shows which symbols will have those errors. Since all of these errors are injected after symbols pass through the PIPE, the symbol values shown in the symbol log are the original, pre-error values.
- When the link is operating with 128b/130b encoding, on the TX side it indicates which symbols are the start of TLP or DLLP tokens.

If symbol logging is enabled then by default the VIP will open and write to the file "symbol_log". To change the file used for symbol logging, use this command:

```
  $msglog_control(MSGLOG_CONTROL_SET_SYMBOL_LOG_FILE, 0, "logfile" );
```

The logfile may be either a relative or full-path name filename.

There are two ways to terminate symbol logging before the end of simulation:

- You can set ENABLE_SYMBOL_LOG to 0.
- Alternatively, you can set the symbol log file to "".

If you use the latter mechanism, then to restart logging during the same simulation you must set a non-null file name.

The following example shows the header of the symbol log and the first few lines in an example with two instances of the VIP.

```
#=======================================================================================
# Symbol logging is performed at the PIPE interface.  If the pipe interface is
# configured as multiple bytes, all bytes transferred at a timestep are logged at the
# same timestep.
#
# Symbol logging uses the following special encodings for all link data rates:
#    z   - Electrical idle
#    ?   - Invalid or unknown value
#    .   - No information available to log.  This may occur at startup, at changes to
# link speed or link width, or if the Rx and Tx sides are operating at offset timesteps
# at either 2.5 or 5 Gt/s.
#    q   - Error injection pending: appended on each symbol that will have disparity
# inverted. Only applies on TX lanes.
#    j   - Error injection pending: appended on each symbol that will have a random bit
# flipped. Only applies on TX lanes.
#    v   - Error injection pending: appended on each symbol that will have an invalid
# encoding. Only applies on TX lanes.
#
# For link operation at 8 Gt/s, the symbols after sync headers are prepended with these
# encodings of the sync headers:
#    @   - 2'b'00 (reserved)
#    *   - 2'b'01 (OS block)
#    =   - 2'b'10 (Data block)
#    $   - 2'b'11 (reserved)
#
# Other special character encodings for link operation at 8 Gt/s:
#    ::  - Data skip cycle (no valid data)
#    +   - Start of  TLP Token. Prepended on 1st symbol of token. Replaces = symbol at
# start of block. Only noted on TX lanes.
#    ^   - Start of DLLP Token. Prepended on 1st symbol of token. Replaces = symbol at
# start of block. Only noted on TX lanes.
#=======================================================================================

     TIME     INSTANCE   R00  |  T00  -> LTSSM State
-----------------------------------------------------
       5:        root0    .   |   z   -> initializing
       5:    endpoint0    .   |   z   -> initializing
       9:        root0    z   |   z   -> initializing
       9:    endpoint0    z   |   z   -> initializing
      13:        root0    z   |   z   -> initializing
      13:    endpoint0    z   |   z   -> initializing
```

10 Functional Coverage

The PCIe VIP provides callback routines which users can utilize for functional coverage. The callbacks are called inside of a class which can easily be extended by users to meet their specific needs. A set of default covergroups is provided, which you can use some or all of.

10.1 Enabling Functional Coverage

To enable functional coverage the following define must be added to the compile line:

+define+PCIESVC_INCLUDE_COVERAGE

+define+PCIESVC_FLAT_INCLUDES

In addition, the file Include/pciesvc_coverage_pkg.sv must be added to the compilation file list. If UVM is being used, pciesvc_coverage_pkg.sv should be listed after the UVM package.

10.2 Class Structure and Callbacks

The classes described in this section are unencrypted and can be viewed in Include/pciesvc_coverage_pkg.sv.

All of the functional coverage classes have an abstract base class(ending is _base) which contains the variables which are to be used by coverage groups as well as pure virtual declarations for Update() and Sample() routines. The Update() callback routines are used to unpack data passed in the callback function into class variables. The Sample() callbacks are used to trigger the coverage groups.

Derived from each base class is a data class where the implementation for all of the Update() tasks is defined. Users that do not wish to use any of the provided functional coverage can extend their own coverage class from the data class.

A functional coverage class is extended from the data class, and in the functional coverage class there is an implementation of the Sample() callbacks along with a number of different functional coverage groups. Users that wish to utilize some or all of the provided functional coverage but modify or add to the existing coverage should extend from the functional coverage classes. Individual covergroups and/or coverpoints can be turned off/adjusted by using standard systemverilog syntax such as option.weight. Please refer to the Systemverilog LRM for more details.

For users who wish to modify the Update() implementation in the _data class, it is recommended to call super.Update() in the child implementation to ensure that the data is unpacked correctly.

10.3 Overriding the Default Coverage Class

Each layer in the VIP protocol stack has a pointer to the corresponding coverage class . The Physical Layer has a pointer to both phy coverage as well as pipe coverage. Any class which replaces the default coverage class must have the _data class for the appropriate layer as its parent.

10.3.1 Overriding With UVM

UVM users should override the default coverage class by using the factory to replace the _data class with the desired class, as shown in the following example:

```
factory.set_type_override_by_type(
   pciesvc_coverage_pkg::link_fc_data::get_type(),
   a_different_coverage_class::get_type(),
   1
);
```

10.3.2 Overriding for SystemVerilog Users

There is an override function for each of the four types of coverage classes: tl, link, phy and pipe. The transaction override function is in the Transaction Layer, the link override function is in the Data Link Layer, and the phy and pipe functions are in the Physical Layer. You must first instantiate the class that you want to use for coverage and then call `new()` on it. Once the class has been constructed the object handle is passed through the override function call. All override function calls are described in Table 10-1. These tasks may also be called through the SystemVerilog API.

Table 10-1 Transaction override functions

Function Name	Arguments	Layer
SetTransactionCoverageClass	new_class – handle to the new coverage class. The new class must be derived from pciesvc_tl_fc_data or pciesvc_tl_fc_coverage.	*SVC_PATH*.port0.tl0
SetLinkCoverageClass	new_class – handle to the new coverage class. The new class must be derived from pciesvc_link_fc_data or pciesvc_link_fc_coverage.	*SVC_PATH*.port0.dl0
SetPhyCoverageClass	new_class – handle to the new coverage class. The new class must be derived from pciesvc_phy_fc_data or pciesvc_phy_fc_coverage.	*SVC_PATH*.port0.phy0
SetPipeCoverageClass	new_class – handle to the new coverage class. The new class must be derived from pciesvc_pipe_fc_data or pciesvc_pipe_fc_coverage.	*SVC_PATH*.port0.phy0

10.4 Transaction Layer

All methods and variables are declared in pciesvc_tl_fc_base, which is loacated in Include/pciesvc_coverage_pkg.sv. Implementation of the Update() functions is in the pciesvc_tl_fc_data class. The covergroups and implementation of the sample() functions are provided in the class pciesvc_tl_fc_coverage.

10.4.1 Transaction Layer Functional Coverage

Table 10-2 lists the covergroups, coverpoints and bins present in the Transaction Layer coverage class.

Table 10-2 Covergroups, coverpoints and bins in the Transaction Layer coverage class

Covergroup	Coverpoints	Bins
cg_tx_tc_vc_mapping	cp_tc	tc_0
		tc_1
		tc_2
		tc_3
		tc_4
		tc_5
		tc_6
		tc_7
	cp_vc	vc_0
		vc_1
		vc_2
		vc_3
		vc_4
		vc_5
		vc_6
		vc_7
	cp_tc_cross_vc	All TC and VC combinations

Table 10-2 Covergroups, coverpoints and bins in the Transaction Layer coverage class (Continued)

Covergroup	Coverpoints	Bins
cg_rx_tc_vc_mapping	cp_tc	tc_0
		tc_1
		tc_2
		tc_3
		tc_4
		tc_5
		tc_6
		tc_7
	cp_vc	vc_0
		vc_1
		vc_2
		vc_3
		vc_4
		vc_5
		vc_6
		vc_7
	cp_tc_cross_vc	All TC and VC combinations

10.4.2 Transaction Layer Callbacks

Transaction Layer functional coverage class callbacks and arguments are listed in Table 10-3.

Table 10-3 Transaction Layer functional coverage class callbacks and arguments

Task Name	Arguments	I/O	Values
UpdateTxTcVcMapping Called every time the Transaction Layer passes a packet to the link.	tx_tc	I	Traffic class of the transmitted TLP
	tx_vc	I	VC which maps to the traffic class of the transmitted TLP
UpdateRxTcVcMapping Called every time the Transa ction Layer receives a packet from the DL.	rx_tc	I	Traffic class of the received TLP
	rx_vc	I	VC which maps to the traffic class of the received TLP
SampleTxTcVcMapping Called immediately following UpdateTxTcVcMapping()	N/A	N/A	N/a
SampleRxTcVcMapping Called immediately following UpdateRxTcVcMapping()	N/A	N/A	N/a

10.5 Data Link Layer

All methods and class variables are declared in pciesvc_link_fc_base, which is located in Include/pciesvc_coverage_pkg.sv. Implementation of the Update() functions is in the pciesvc_link_fc_data class. The covergroups and implementation of the sample() functions are provided in the pciesvc_link_fc_coverage class.

Please note for the TLP and DLLP Update tasks that all fields in the argument list may not be valid depending on the type of packet. For example, the message_code field is valid only for message TLPs, and should be disregarded on other TLPs. The provided covergroups cover most aspects of TLP/DLLP transmission, but not all TLP fields have a coverpoin. However, all TLP fields are updated during the UpdateTxTLP/UpdateRxTLP callbacks so that users can create their own coverage if necessary.

10.5.1 Data Link Layer Functional Coverage

Table 10-4 lists the covergroups, coverpoints and bins present in the Data Link Layer layer coverage class. Note that the type field for TLPs and DLLPs is sampled in several coverpoints. Having different types of TLPs in different coverpoints allows users to easily change weights/goals within the coverpoints to cover only the types of packets they are interested in.

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class

Covergroup	Coverpoint	Bins	Comment
cg_tx_dllp	cp_dllp_type_acknak	ACK	ACK/NAK DLLPs
		NAK	
	cp_dllp_type_pm	PM Enter L1	Power Management DLLPS
		PM Enter L23	
		PM Active State Request	
		PM Request Ack	
	cp_dllp_type_vendor_specific	Vendor Specific	Vendor Specific
	cp_dllp_type_fc_vc0	initfc1_p_vc0	VC0 Flow Control DLLPs
		initfc1_np_vc0	
		initfc1_cpl_vc0	
		initfc2_p_vc0	
		initfc2_np_vc0	
		initfc2_cpl_vc0	
		updatefc_p_vc0	
		updatefc_np_vc0	
		updatefc_cpl_vc0	

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_dllp_type_fc_vc1	initfc1_p_vc1	VC1 Flow Control DLLPs
		initfc1_np_vc1	
		initfc1_cpl_vc1	
		initfc2_p_vc1	
		initfc2_np_vc1	
		initfc2_cpl_vc1	
		updatefc_p_vc1	
		updatefc_np_vc1	
		updatefc_cpl_vc1	
	cp_dllp_type_fc_vc2	initfc1_p_vc2	VC2 Flow Control DLLPs
		initfc1_np_vc2	
		initfc1_cpl_vc2	
		initfc2_p_vc2	
		initfc2_np_vc2	
		initfc2_cpl_vc2	
		updatefc_p_vc2	
		updatefc_np_vc2	
		updatefc_cpl_vc2	
	cp_dllp_type_fc_vc3	initfc1_p_vc3	VC3 Flow Control DLLPs
		initfc1_np_vc3	
		initfc1_cpl_vc3	
		initfc2_p_vc3	
		initfc2_np_vc3	
		initfc2_cpl_vc3	
		updatefc_p_vc3	
		updatefc_np_vc3	
		updatefc_cpl_vc3	

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_dllp_type_fc_vc4	initfc1_p_vc4	VC4 Flow Control DLLPs
		initfc1_np_vc4	
		initfc1_cpl_vc4	
		initfc2_p_vc4	
		initfc2_np_vc4	
		initfc2_cpl_vc4	
		updatefc_p_vc4	
		updatefc_np_vc4	
		updatefc_cpl_vc4	
	cp_dllp_type_fc_vc5	initfc1_p_vc5	VC5 Flow Control DLLPs
		initfc1_np_vc5	
		initfc1_cpl_vc5	
		initfc2_p_vc5	
		initfc2_np_vc5	
		initfc2_cpl_vc5	
		updatefc_p_vc5	
		updatefc_np_vc5	
		updatefc_cpl_vc5	

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_dllp_type_fc_vc6	initfc1_p_vc6	VC6 Flow Control DLLPs
		initfc1_np_vc6	
		initfc1_cpl_vc6	
		initfc2_p_vc6	
		initfc2_np_vc6	
		initfc2_cpl_vc6	
		updatefc_p_vc6	
		updatefc_np_vc6	
		updatefc_cpl_vc6	
	cp_dllp_type_fc_vc7	initfc1_p_vc7	VC7 Flow Control DLLPs
		initfc1_np_vc7	
		initfc1_cpl_vc7	
		initfc2_p_vc7	
		initfc2_np_vc7	
		initfc2_cpl_vc7	
		updatefc_p_vc7	
		updatefc_np_vc7	
		updatefc_cpl_vc7	
	cp_hdr_fc	less_8	HDR FC Value (sampled only on flow control type DLLPs)
		less_32	
		less_128	
		less_255	
	cp_data_fc	less_128	DATA FC Value (sampled only on flow control type DLLPs)
		less_512	
		less_1024	
		less_4096	
	cp_hdr_cross_fc_vc0	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc0	hdr cross flow control type for VC0
	cp_hdr_cross_fc_vc1	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc1	hdr cross flow control type for VC1
	cp_hdr_cross_fc_vc2	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc2	hdr cross flow control type for VC2

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_hdr_cross_fc_vc3	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc3	hdr cross flow control type for VC3
	cp_hdr_cross_fc_vc4	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc4	hdr cross flow control type for VC4
	cp_hdr_cross_fc_vc5	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc5	hdr cross flow control type for VC5
	cp_hdr_cross_fc_vc6	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc6	hdr cross flow control type for VC6
	cp_hdr_cross_fc_vc7	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc7	hdr cross flow control type for VC7
	cp_data_cross_fc_vc0	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc0	data cross flow control type for VC0
	cp_data_cross_fc_vc1	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc1	data cross flow control type for VC1
	cp_data_cross_fc_vc2	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc2	data cross flow control type for VC2
	cp_data_cross_fc_vc3	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc3	data cross flow control type for VC3
	cp_data_cross_fc_vc4	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc4	data cross flow control type for VC4

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_data_cross_fc_vc5	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc5	data cross flow control type for VC5
	cp_data_cross_fc_vc6	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc6	data cross flow control type for VC6
	cp_data_cross_fc_vc7	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc7	data cross flow control type for VC7
	cp_dllp_error_injections	corrupt_crc	Error injections for transmitted DLLPs
		unknown_type	
		rsvd_non_zero	
		duplicate_ack	
		missing_start	
		missing_end	
		corrupt_disparity	
		code_violation	
cg_rx_dllp	cp_dllp_type_acknak	ACK NAK	ACK/NAK DLLPs
	cp_dllp_type_pm	PM Enter L1 PM Enter L23	Power Management DLLPS
		PM Active State Request	
		PM Request Ack	
	cp_dllp_type_vendor_specific	Vendor Specific	Vendor Specific

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_dllp_type_fc_vc0	initfc1_p_vc0	VC0 Flow Control DLLPs
		initfc1_np_vc0	
		initfc1_cpl_vc0	
		initfc2_p_vc0	
		initfc2_np_vc0	
		initfc2_cpl_vc0	
		updatefc_p_vc0	
		updatefc_np_vc0	
		updatefc_cpl_vc0	
	cp_dllp_type_fc_vc1	initfc1_p_vc1	VC1 Flow Control DLLPs
		initfc1_np_vc1	
		initfc1_cpl_vc1	
		initfc2_p_vc1	
		initfc2_np_vc1	
		initfc2_cpl_vc1	
		updatefc_p_vc1	
		updatefc_np_vc1	
		updatefc_cpl_vc1	
	cp_dllp_type_fc_vc2	initfc1_p_vc2	VC2 Flow Control DLLPs
		initfc1_np_vc2	
		initfc1_cpl_vc2	
		initfc2_p_vc2	
		initfc2_np_vc2	
		initfc2_cpl_vc2	
		updatefc_p_vc2	
		updatefc_np_vc2	
		updatefc_cpl_vc2	

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_dllp_type_fc_vc3	initfc1_p_vc3	VC3 Flow Control DLLPs
		initfc1_np_vc3	
		initfc1_cpl_vc3	
		initfc2_p_vc3	
		initfc2_np_vc3	
		initfc2_cpl_vc3	
		updatefc_p_vc3	
		updatefc_np_vc3	
		updatefc_cpl_vc3	
	cp_dllp_type_fc_vc4	initfc1_p_vc4	VC4 Flow Control DLLPs
		initfc1_np_vc4	
		initfc1_cpl_vc4	
		initfc2_p_vc4	
		initfc2_np_vc4	
		initfc2_cpl_vc4	
		updatefc_p_vc4	
		updatefc_np_vc4	
		updatefc_cpl_vc4	
	cp_dllp_type_fc_vc5	initfc1_p_vc5	VC5 Flow Control DLLPs
		initfc1_np_vc5	
		initfc1_cpl_vc5	
		initfc2_p_vc5	
		initfc2_np_vc5	
		initfc2_cpl_vc5	
		updatefc_p_vc5	
		updatefc_np_vc5	
		updatefc_cpl_vc5	

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_dllp_type_fc_vc6	initfc1_p_vc6	VC6 Flow Control DLLPs
		initfc1_np_vc6	
		initfc1_cpl_vc6	
		initfc2_p_vc6	
		initfc2_np_vc6	
		initfc2_cpl_vc6	
		updatefc_p_vc6	
		updatefc_np_vc6	
		updatefc_cpl_vc6	
	cp_dllp_type_fc_vc7	initfc1_p_vc7	VC7 Flow Control DLLPs
		initfc1_np_vc7	
		initfc1_cpl_vc7	
		initfc2_p_vc7	
		initfc2_np_vc7	
		initfc2_cpl_vc7	
		updatefc_p_vc7	
		updatefc_np_vc7	
		updatefc_cpl_vc7	
	cp_hdr_fc	less_8	HDR FC Value (sampled only on flow control type DLLPs)
		less_32	
		less_128	
		less_255	
	cp_data_fc	less_128	DATA FC Value (sampled only on flow control type DLLPs)
		less_512	
		less_1024	
		less_4096	
	cp_hdr_cross_fc_vc0	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc0	hdr cross flow control type for VC0

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_hdr_cross_fc_vc1	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc1	hdr cross flow control type for VC1
	cp_hdr_cross_fc_vc2	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc2	hdr cross flow control type for VC2
	cp_hdr_cross_fc_vc3	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc3	hdr cross flow control type for VC3
	cp_hdr_cross_fc_vc4	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc4	hdr cross flow control type for VC4
	cp_hdr_cross_fc_vc5	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc5	hdr cross flow control type for VC5
	cp_hdr_cross_fc_vc6	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc6	hdr cross flow control type for VC6
	cp_hdr_cross_fc_vc7	all combinations of cp_hdr_fc bins and cp_dllp_type_fc_vc7	hdr cross flow control type for VC7
	cp_data_cross_fc_vc0	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc0	data cross flow control type for VC0
	cp_data_cross_fc_vc1	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc1	data cross flow control type for VC1
	cp_data_cross_fc_vc2	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc2	data cross flow control type for VC2
	cp_data_cross_fc_vc3	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc3	data cross flow control type for VC3
	cp_data_cross_fc_vc4	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc4	data cross flow control type for VC4

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_data_cross_fc_vc5	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc5	data cross flow control type for VC5
	cp_data_cross_fc_vc6	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc6	data cross flow control type for VC6
	cp_data_cross_fc_vc7	all combinations of cp_data_fc bins and cp_dllp_type_fc_vc7	data cross flow control type for VC7
cg_tx_tlp	cp_mem_requests	mem_rd_req_32	fmt/type for Memory Requests
		mem_rd_req_64	
		mem_wr_req_32	
		mem_wr_req_64	
	cp_mem_rd_lk_requests	mem_rd_req_lk_32	fmt/type for Mem Read Locked Requests
		mem_rd_req_lk_64	
	cp_io_requests	io_rd_req	fmt/type for I/O Requests
		io_wr_req	
	cp_cfg_type0_requests	cfg_rd_req0	fmt/type for Type 0 Config Requests
		cfg_wr_req0	
	cp_cfg_type1_requests	cfg_rd_req1	fmt/type for Type 1 Config Requests
		cfg_wr_req01	
	cp_msg	Msg	fmt/type for Message TLPs
		MsgD	
	cp_cpl	Cpl	fmt/type for Completion TLPs
		CplD	
	cp_lk_cpl	CplLk	fmt/type for Completion Locked TLPs
		CplDLk	

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_atomic_ops	FetchAdd32	fmt/type for Atomic Ops
		FetchAdd64	
		Swap	
		Swap64	
		CAS32	
		CAS64	
	cp_traffic class	tc0	Traffic Class
		tc1	
		tc2	
		tc3	
		tc4	
		tc5	
		tc6	
		tc7	
	cp_transaction_hint	0/1	Transaction hint
	cp_tlp_digest	0/1	TLP digest bit
	cp_error_poison	0/1	Error Poison bit
	cp_address_transalation	default_untranslated	Address transaction bit
		translation_request	
		translated	
	cp_length	length_1	length field
		length_2_thru_1023	
		length_1024	
	cp_attr_id_order	0/1	ID ordering attribute bit
	cp_attr_relax_order	0/1	relaxed ordering attribute bit
	cp_attr_no_snoop	0/1	nosnoop attribute bit

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_first_dw_be	autobins for 4’b000-4’b1111	First DW byte enable. Fires only on mem, I/O and CFG type TLPs
	cp_last_dw_be	autobins for 4'b0-4'b1111	Last DW byte enable. Fires only on mom, I/O and CFG type TLPs
	cp_ph	0/1	processing hint
	cp_msg_code	cp_assert_inta	Message Code Type
		cp_assert_intb	
		cp_assert_intc	
		cp_assert_intd	
		cp_deassert_inta	
		cp_deassert_intb	
		cp_deassert_intc	
		cp_deassert_intd	
		pm_active_state_nak	
		pm_pme	
		pm_pme_turn_off	
		pm_pme_to_ack	
		err_cor	
		err_non_fatal	
		err_fatal	
		unlock	
		set_slot_power_limit	
		OBFF	
	cp_completion_status	successful completion	Completion Status
		unsupported request	
		completer abort	
		CRS	

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_sequence_num	seq_num_0	TLP Sequence Number
		seq_num_1_thru_4095	
		seq_num_4095	
	cp_ei_code	ei_none	Error Injection Codes
		ei_corupt_crc	
		ei_illegal_seq_num	
		ei_duplicate_seq_num	
		ei_nullified	
		ei_nullified_good_lcrc	
		ei_nullified_corrupt_lcrc	
		ei_corrupt_disparity	
		ei_code_violation	
		ei_missing_start	
		ei_missing_end	
		ei_8g_corrupt_header_crc	
		ei_8g_corrupt_header_parity	
		ei_corrupt_ecrc	
		ei_ignore_credit	
		ei_expect_ur	
		ei_expect_crs	
		ei_expect_ca	
		ei_expect_timeout	
cg_rx_tlp	cp_mem_requests	mem_rd_req_32	fmt/type for Memory Requests
		mem_rd_req_64	
		mem_wr_req_32	
		mem_wr_req_64	

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_mem_rd_lk_requests	mem_rd_req_lk_32	fmt/type for Mem Read Locked Requests
		mem_rd_req_lk_64	
	cp_io_requests	io_rd_req	fmt/type for I/O Requests
		io_wr_req	
	cp_cfg_type0_requests	cfg_rd_req0	fmt/type for Type 0 Config Requests
		cfg_wr_req0	
	cp_cfg_type1_requests	cfg_rd_req1	fmt/type for Type 1 Config Requests
		cfg_wr_req01	
	cp_msg	Msg	fmt/type for Message TLPs
		MsgD	
	cp_cpl	Cpl	fmt/type for Completion TLPs
		CplD	
	cp_lk_cpl	CplLk	fmt/type for Completion Locked TLPs
		CplDLk	
	cp_atomic_ops	FetchAdd32	fmt/type for Atomic Ops
		FetchAdd64	
		Swap	
		Swap64	
		CAS32	
		CAS64	
	cp_traffic class	tc0	Traffic Class
		tc1	
		tc2	
		tc3	
		tc4	
		tc5	
		tc6	
		tc7	

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_th	0/1	Transaction hint
	cp_td	0/1	TLP digest bit
	cp_ep	0/1	Error Poison bit
	cp_at	0/1	Address transaction bit
	cp_length	length_1 length_2_thru_1023	length field
		length_1024	
	cp_attr_id_order	0/1	ID ordering attribute bit
	cp_attr_relax_order	0/1	relaxed ordering attribute bit
	cp_attr_no_snoop	0/1	nosnoop attribute bit
	cp_first_dw_be	autobins for 4'b000-4'b1111	First DW byte enable. Fires only on mem, I/O and CFG type TLPs
	cp_last_dw_be	autobins for 4'b0-4'b1111	Last DW byte enable. Fires only on mom, I/O and CFG type TLPs
	cp_ph	0/1	processing hint

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
	cp_msg_code	cp_assert_inta	Message Code Type
		cp_assert_intb	
		cp_assert_intc	
		cp_assert_intd	
		cp_deassert_inta	
		cp_deassert_intb	
		cp_deassert_intc	
		cp_deassert_intd	
		pm_active_state_nak	
		pm_pme	
		pm_pme_turn_off	
		pm_pme_to_ack	
		err_cor	
		err_non_fatal	
		err_fatal	
		unlock	
		set_slot_power_limit	
		OBFF	
	cp_completion_status	successful completion	Completion Status
		unsupported request	
		completer abort	
		CRS	
	cp_sequence_num	seq_num_0	Sequence number assigned to TLP
		seq_num_1_thru_4094	
		seq_num_4095	
cg_tx_ipg	cp_tx_ipg	ipg_0	Inter packet gap of transmitted packets
		ipg_1	
		ipg_2	
		ipg_3_to_4	
		ipg_5_to_8	
		ipg_9_to_16	
		ipg_16_to_32	
		ipg_greater_than_32	

Table 10-4 Covergroups, coverpoints and bins in the Data Link Layer layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
cg_rx_ipg	cp_rx_ipg	ipg_0	Inter packet gap of received packets
		ipg_1	
		ipg_2	
		ipg_3_to_4	
		ipg_5_to_8	
		ipg_9_to_16	
		ipg_16_to_32	
		ipg_greater_than_32	

10.5.2 Link Layer Callbacks

Data Link Layer callbacks are listed in Table 10-5.

Table 10-5 Data Link Layer callbacks

Function Name	Arguments	I/O	Values
UpdateTxDLLP This function is called every time the link finishes sending a DLLP.	dllp[63:0]	I	64 bit array containing the DLLP data
	ei_code[31:0]	I	Error injection code associated with the DLLP.
UpdateRxDLLP Called every time the link received a DLLP.	dllp[63:0]	I	64 bit array containing the DLLP data
	rx_status[31:0]	I	Status of the received DLLP. Status bits are defined in Include/pciesvc_parms.v under RECEIVED_TLP_STATUS*

Table 10-5 Data Link Layer callbacks (Continued)

Function Name	Arguments	I/O	Values
UpdateTxTLP Called after the link sends the last byte of a TLP.	tlp_fmt[2:0]	I	Format field
	tlp_type[4:0]	I	Type field
	tc[2:0]	I	Traffic class
	th	I	Transaction hint
	td	I	TLP Digest bit
	ep	I	Error/Poison bit
	attr_id_order	I	ID Order attribute bit
	attr_relax_order	I	Relaxed order attribute bit
	attr_no_snoop	I	No snoop attribute attribute
	at[1:0]	I	address translation
	length[9:0]	I	length field
	ecrc[31:0]	I	ECRC(digest) value
	lcrc[31:0]	I	Link CRC value
	sequence_num (int)	I	Sequence number of the TLP
	requester_id[15:0]	I	Requester ID field
	tag[7:0]	I	Tag value
	first_dw_be[3:0]	I	First DW byte enable field.
	last_dw_be[3:0]	I	Last DW byte enable field.
	address[63:0]	I	Address field.

Table 10-5 Data Link Layer callbacks (Continued)

Function Name	Arguments	I/O	Values
	ph[1:0]	I	Processing hint
	bus_num[7:0]	I	bus number
	device_num[2:0]	I	device number
	function_num[2:0]	I	function number
	register_num[9:0]	I	Combines reg and ext_reg field for config TLPs
	message_code[7:0]	I	Message code field
	message_dword2[31:0]	I	2nd dword of message TLP. This would be used for vendor specific messages.
	message_dword3[31:0]	I	3rd dword of message TLP.
	completer_id[15:0]	I	Completer ID field
	completion_status[2:0]	I	Completion status
	bcm	I	Byte count modified field
	byte_count[11:0]	I	Byte count field
	lower_address[6:0]	I	Lower address field.
	steering_tag[7:0]	I	Steering tag.
	payload_data (dynamic array)	I	Payload data, if any present.
	ei_code [31:0]	I	Error injection code associated with the TLP.
UpdateRxTLP Called after the link receives the last byte of a TLP.	tlp_fmt[2:0]	I	Format field
	tlp_type[4:0]	I	Type field
	tc[2:0]	I	Traffic class
	th	I	Transaction hint
	td	I	TLP Digest bit
	ep	I	Error/Poison bit
	attr_id_order	I	ID Order attribute bit
	attr_relax_order	I	Rleaxed order attribute bit
	attr_no_snoop	I	No snoop attribute attribute

Table 10-5 Data Link Layer callbacks (Continued)

Function Name	Arguments	I/O	Values
	at[1:0]	I	address translation
	length[9:0]	I	length field
	ecrc[31:0]	I	ECRC(digest) value
	lcrc[31:0]	I	Link CRC value
	sequence_num (int)	I	Sequence number of the TLP
	requester_id[15:0]	I	Requester ID field
	tag[7:0]	I	Tag value
	first_dw_be[3:0]	I	First DW byte enable field.
	last_dw_be[3:0]	I	Last DW byte enable field.
	address[63:0]	I	Address field.
	ph[1:0]	I	Processing hint
	bus_num[7:0]	I	bus number
	device_num[2:0]	I	device number
	function_num[2:0]	I	function number
	register_num[9:0]	I	Combines reg and ext_reg field for config TLPs
	message_code[7:0]	I	Message code field
	message_dword2[31:0]	I	2nd dword of message TLP. This would be used for vendor specific messages.
	message_dword3[31:0]	I	3rd dword of message TLP.
	completer_id[15:0]	I	Completer ID field
	completion_status[2:0]	I	Completion status
	bcm	I	Byte count modified field
	byte_count[11:0]	I	Byte count field
	lower_address[6:0]	I	Lower address field.I
	steering_tag[7:0]	I	Steering tag.
	payload_data (dynamic array)	I	Payload data, if any present.
UpdateTxIpg Called every time a new packet starts transmission.	ipg(int)	I	Number of bytes between current packet and previously transmitted packet.
UpdateRxIpg Called on the start of a new received packet.	ipg(int)	I	Number of bytes between current packet and previously received packet

Table 10-5 Data Link Layer callbacks (Continued)

Function Name	Arguments	I/O	Values
UpdateDLCMSMState This function is called every time the DLCMSM changes state.	state[31:0]	I	Current state of the DLCMSM (states are defined in Verilog/Link_Layer/ pciesvc_ll_parms.v)
UpdateFCState Called every time the FC state machine changes state.	state[31:0]	I	Current state of the flow control state machine. States are defined in Verilog/Link_Layer/ pciesvc_ll_parms.v
SampleTxDLLP Called immediately after UpdateTxDLLP()	n/a	n/a	n/a
SampleRxDLLP Called immediately after UpdateRxDLLP()	n/a	n/a	n/a
SampleTxTLP Called immediately after UpdateTxTLP()	n/a	n/a	n/a
SampleRxTLP Called immediately after SampleRxTLP	n/a	n/a	n/a
SampleTxIPG Called immediately after UpdateTxIPG()	n/a	n/a	n/a
SampleRxIPG Called immediately following SampleRxIPG	n/a	n/a	n/a
SampleDLCMSMState Called immediately following UpdateDLCMSMState()	n/a	n/a	n/a
SampleFCState Called immediately following UpdateFcState()	n/a	n/a	n/a

10.6 Physical Layer

All methods and class variables are declared in pciesvc_phy_fc_base, which is located in Include/ pciesvc_coverage_pkg.sv. Implementation of the Update() functions is in the pciesvc_phy_fc_data class. The covergroups and implementation of the sample() functions are provided in the class pciesvc_phy_fc_coverage.

A covergroup with all of the legal transitions in the LTSSM has been provided, though users should note that the PCIESVC LTSSM hitting a certain state doesn't necessarily imply that the DUT has successfully entered that state. Depending on whether or not the VIP is upstream or downstream not all state transitions may apply. Finally, in many cases there are multiple conditions which may trigger a transition from one state to the next (example: transitioning from L0 to recover due to receiving a training set, or going from L0 to recover for a speed change). Additional coverage will be required to capture these conditions.

10.6.1 Physical Layer Functional Coverage

Covergroups, coverpoints and bins in the Physical Layer coverage class are described in Table 10-6.

Table 10-6 Covergoups, coverpoints and bins in the Physical Layer coverage class

Covergroup	Coverpoint	Bins	Comment
cg_negotiated_data_rate	cp_negotiated_data_rate	speed_2_5G	Data rate upon entry into L0
		speed_5_0G	
		speed_8_0G*	
cg_negotiated_link_width	cp_negotiated_link_width	link_width_1	Link width upon entry into L0
		link_width_2	
		link_width_4	
		link_width_8	
		link_width_12	
		link_width_16	
		link_width_32	

Table 10-6 Covergoups, coverpoints and bins in the Physical Layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
cg_ltssm_state_transitions cp_ltss_state_transitions		detect_quiet_to_detect_active	State transitions of all LTSSM states except for the L0s substates, which have their own separate coverage.
		detect_active_to_polling_active	
		polling_active_to_polling_compliance	
		polling_active_to_polling_configuration	
		polling_active_to_detect_quiet	
		polling_compliance_to_detect_quiet	
		polling_compliance_to_polling_active	
		polling_configuration_to_configuration_ linkwidth_start	
		polling_configuration_to_detect_quiet	
		configuration_linkwidth_start_to_ disabled	
		configuration_linkwidth_start_to_l oopback_entry	
		configuration_linkwidth_start_to_ configuration_linkwidth_accept	
		configuration_linkwidth_start_to_detect _quiet	
		configuration_linkwidth_accept_to_ configuation_lanenum_wait	
		configuration_linkwidth_accept_to_ detect_quiet	
		configuration_lanenum_accept_to_ configuration_complete	

Table 10-6 Covergoups, coverpoints and bins in the Physical Layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
		configuration_lanenum_accept_to_configuration_lanenum_wait	
		configuration_lanenum_accept_to_detect_quiet	
		configuration_lanenum_wait_to_configuration_lanenum_accept	
		configuration_lanenum_wait_to_detect_quiet	
		configuration_complete_to_configuration_idle	
		configuration_complete_to_detect_quiet	
		configuration_idle_to_l0	
		configuration_idle_to_detect_quiet	
		configuration_idle_to_recovery_rcvrlock	
		recovery_rcvrlock_to_recovery_equalization_phase0*	
		recovery_rcvrlock_to_recover_equalization_phase1*	
		recovery_rcvrlock_to_recovery_rcvrcfg	
		recovery_rcvrlock_to_recovery_speed	
		recovery_rcvrlock_to_configuration_linkwidth_start	
		recovery_rcvrlock_to_detect_quiet	
		recovery_equalization_phase0_to_recovery_speed*	
		recovery_equalization_phase0_to_recovery_equalization_phase1	
		recovery_equalization_phase1_to_recovery_rcvrlock*	
		recovery_equalization_phase1_to_recovery_speed*	

Table 10-6 Covergoups, coverpoints and bins in the Physical Layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
		recovery_equalization_phase2_to_ recovery_speed*	
		recovery_equalization_phase2_to_ recovery_equaliztion_phase3*	
		recovery_equalization_phase3_to_ recovery_speed*	
		recovery_equalization_phase3_to_ recovery_rcvrlock*	
		recovery_speed_to_recovery_rcvrlock	
		recovery_rcvrcfg_to_recovery_idle	
		recovery_rcvrcfg_to_configuration_link width_start	
		recovery_rcvrcfg_to_recovery_idle	
		recovery_rcvrcfg_to_configuration_link width_start	
		recovery_rcvrcfg_to_detect_quiet	
		recovery_idle_to_disabled	
		recovery_idle_to_hot_reset	
		recovery_idle_to_configuration_ linkwidth_start	
		recovery_idle_to_loopback_entry	
		recovery_idle_to_l0	
		recovery_idle_to_detect_quiet	
		recovery_idle_to_recovery_rcvrlock	
		l0_to_recovery_rcvrlock	
		l0_to_l1_entry	
		l1_entry_to_l1_idle	
		l1_entry_to_recovery_rcvrlock	
		l1_idle_to_l1_1_idle*	

Table 10-6 Covergoups, coverpoints and bins in the Physical Layer coverage class (Continued)

Covergroup	Coverpoint	Bins	Comment
		l1_idle_to_l1_2_entry*	
		l1_2_entry_to_l1_2_idle*	
		l1_2_idle_to_l1_2_exit*	
		l1_2_exit_to_l1_idle*	
		l1_1_idle_to_l1_idle*	
		l1_1_idle_to_recovery_rcvrlock*	
		l0_to_l2_idle	
		l2_idle_to_detect_quiet	
		disabled_to_detect_quiet	
		loopback_entry_to_loopback_active	
		loopback_entry_to_loopback_exit	
		loopback_active_to_loopback_exit	
		loopback_exit_to_detect_quiet	
		hot_reset_to_detect_quiet	
cg_tx_l0s_substate_ transitions	cp_tx_l0s_substate	l0_to_l0s_entry	L0s substate transitions for the transmit side
		l0s_entry_to_l0s_idle	
		l0s_idle_to_l0s_fts	
		l0s_fts_to_l0	
cg_rx_l0s_substate_ transitions	cp_rx_l0s_substate_ transitions	l0_to_l0s_entry	L0s substate transitions for the receive side
		l0s_entry_to_l0s_idle	
		l0s_idle_to_l0s_fts	
		l0s_fts_to_l0	
		los_fts_to_recovery_rcvrlock	
*Exist for 8G models only			

10.6.2 Physical Layer Callbacks

Callbacks in the Physical Layer are defined in Table 10-7.

Table 10-7 Callbacks in the Physical Layer

Function Name	Arguments	I/O	Values
UpdateNegotiatedDataRate Called every time the LTSSM enters the L0 state.	rate [2:0]	I	Pipe rate value upon entering L0
UpdateNegotiatedLinkWidth Called every time the LTSSM enters the L0 state.	width (int)	I	Link width upon entering L0
UpdateLtssmState Called every time the LTSSM enters a new state.	state [31:0]	I	Current LTSSM state
UpdateTxL0sSubstate Called every time the LTSSM transmit side enters a new L0s substate.	tx_l0s_substate[31:0]	I	Current LTSSM tx substate
UpdateRxL0sSubstate Called every time the LTSSM receive side enters a new L0s substate.	rx_l0s_substate[31:0]	I	Current LTSSM rx substate
SampleNegotiatedDataRate Called immediately following UpdateNegotiatedDataRate()	n/a	n/a	n/a
SampleNegotiatedLinkWidth Called immediately following UpdateNegotiatedLinkWidth	n/a	n/a	n/a
SampleLtssmState Called immediately following UpdateLtssmState	n/a	n/a	n/a
SampleTxL0sSubstate Called immediately following UpdateTxL0sSubstate	n/a	n/a	n/a
SampleRxL0sSubstate Called immediately following UpdateRxL0sSubstate	n/a	n/a	n/a

10.7 PIPE Interface

All methods and class variables are declared in pciesvc_pipe_fc_base, which is located in Include/pciesvc_coverage_pkg.sv. Implementation of the Update() functions is in the pciesvc_pipe_fc_data class. The covergroups and implementation of the Sample() functions are provided in the class pciesvc_pipe_fc_coverage.

10.7.1 PIPE Functional Coverage

PIPE functional covergroups coverpoints, and bins are listed in Table 10-8.

Table 10-8 PIPE covergroups, coverpoints and bins

Covergroup	Coverpoint	Bins	Comment
cg_rate	cp_rate	PIPE_RATE_2_5G	Valid values for the pipe rate signal
		PIPE_RATE_5G	
		PIPE_RATE_8G*	
cg_powerdown	cp_powerdown	P0	Valid values for the pipe powerdown signal
		P0s	
		P1	
		P2	
data_bus_width	cp_data_bus_width	bus_width_8_bits	Valid values for the data_bus_width signal.
		bus_width_16_bits	
		bus_width_32_bits	

* Exists for 8G models only

10.7.2 PIPE Interface Callbacks

Callbacks in the PIPE interface are listed in Table 10-9.

Table 10-9 PIPE callbacks

Function Name	Arguments	I/O	Values
UpdateRate Called every time the pipe signal rate changes.	rate [1:0]	I	Pipe rate value
UpdataPowerDown Called every time the pipe signal powerdown changes.	powerdown[2:0]	I	Pipe powerdown value
UpdateDataBusWidth This function is called every time the pipe signal data_bus_width changes value.	data_bus_width[1:0]	I	Pipe data bus width value

Table 10-9 PIPE callbacks (Continued)

Function Name	Arguments	I/O	Values
UpdateTxPipeLane This function is called once per lane per pipe clock cycle and copies over all of the tx signals for 1 lane into class variables.	lane_number (int)	I	lane number to be updated
	tx_data[31:0]	I	transmit data
	tx_data_k[3:0]	I	transmit data control bits
	tx_compliance	I	transmit compliance
	tx_data_valid*	I	data_valid
	tx_start_block*	I	start block
	tx_sync_header*	I	transmit sync header
	tx_elec_idle	I	transmit electrical idle
UpdateRxPipeLane This function is called once per lane per pipe clock cycle and copies over all of the rx signals for 1 lane into class variables.	rx_data[31:0]	I	receive data
	rx_data_k[3:0]	I	receive data control bits
	rx_status[1:0]	I	receive status
	rx_valid	I	receive data valid
	rx_elec_idle	I	receive electricle idle
	rx_data_valid*	I	receive data valid
	rx_start_block*	I	received start of a new block
	rx_sync_header*	I	value of sync header
	invert_rx_polarity	I	invert received polarity
SampleRate Called immediately after UpdateRate()	n/a	n/a	n/a
SamplePowerDown Called immediately after SamplePowerDown()	n/a	n/a	n/a
SampleDataBusWidth Called immediately after UpdateDataBusWidth().	n/a	n/a	n/a
SampleTxPipeLane Called once per pipe clock after all tx lanes are updated	n/a	n/a	n/a
SampleRxPipeLane Called once per pipe clock after all rx lanes are updated.	n/a	n/a	n/a
*These signals present in 8G models only			

11
PCIe VIP Verification Plans

VCS Verification Planner is a verification planning system that is incorporated into the Synopsys PCIe VIP. It allows you to apply data from test runs to a verification plan which you can use to track the progress of the verification project.

A verification plan is a way to describe a scheme for achieving verification goals for the PCIe VIP. The verification plan contains feature declarations, attributes, goals, and metrics for analyzing the progress of the verification.

The Synopsys PCIe VIP provides predefined verification plans listed in the tables below. These verification plans are available in the Verilog/Doc/VerificationPlans directory.

11.1 Top-level Plans

Top-level PCIe VIP verification plans are listed in Table 11-1.

Table 11-1 Top-level verification plans for the PCIe VIP

DUT Operation	VIP Operation	Top-Level Plan
Root	Endpoint	pciesvc_root_dut_toplevel_fc_plan.xml
Endpoint	Endpoint	pciesvc_endpoint_dut_toplevel_fc_plan.xml

11.2 Subplans

Subplans of the top-level PCIe VIP verification plan are listed in Table 11-2.

Table 11-2 Subplans of the top-level verification plan for the PCIe VIP

Subplan Name	VIP Layer	Comments
pciesvc_transaction.xml	Transaction	TC/VC Mapping
pciesvc_link.xml	Link	TX/RX packets and their fields
pciesvc_phy.xml	Physical	Link speed, width, LTSSM and PIPE

11.3 Back-annotation with VCS Verification Planner

Back-annotation of a verification plan refers to applying the results of a test to the verification plan. Use the following command to back-annotate the PCIe VIP verification plan:

```
hvp annotate -dir simv.vdb -plan usb_svt/doc/VerificationPlans/
svt_usb_device_dut_ss_toplevel_fc_plan.xml
```

The inputs to the command above are:

- simv.vdb – The coverage database file
- usb_svt/doc/VerificationPlans/svt_usb_device_dut_ss_toplevel_fc_plan.xml – The top-level functional verification plan

12 Debugging the PCIe VIP

The PCIe VIP is implemented as a gray-box SystemVerilog model. Its internal states are identified by strings and can be observed in waveform viewers.

Runtime log files are particularly useful for debugging the PCIe. The model implements the SystemVerilog `$msglog` task to control what information is written to the log during a test run.

The Synopsys Protocol Analyzer also can be used as a debugging tool. The Protocol Analyzer tool provides the following debugging features:

- Protocol-aware GUI
- Viewing of protocol activity, including the ability to
  - View all layers with parent, child, and sibling
  - Browse multiple protocols simultaneously
- Immediate error identification
  - Highlights errors on a protocol-centric view
  - Provides detailed information on demand
  - Offers extensive filtering to speed up debugging
- Unified debugging
  - Synchronized to log files with back annotation of errors into the protocol view
  - Linked to Synopsys VIP user documentation
  - Can support customer VIPs
  - Supports synchronized debugging with DVE and Verdi.

12.1 Using Run Logs for Debugging

All `$msglog` PLI calls print to the standard output (see "PLI Task $msglog" on page 240). Each `$msglog` entry includes the path to the module that initiated the log entry.

12.1.1 Setting Log Severity Levels

The `$msglog` task is used to display and log messages at a severity level you specify. The following are the levels for logging, in order of decreasing severity:

- ERROR
- WARNING
- NOTICE
- INFO
- TRANSACTION
- FSM / FRAME
- DWORD
- DEBUG

ERROR, WARNING, and NOTICE are three levels of critical errors and should always be addressed by the test writer.

12.1.2 Logging Transactions

Transactions are logged in a separate log, with each transaction on one line in the log. You use the `$msglog_control` task to with the MSGLOG_CONTROL_SET_TRANSACTION_LOG_FILE command argument to enable transaction logging. See “The Transaction Logger” on page 183.

12.1.3 Searching the Log

Search for “Queued” in the log file to find TLPs queued to the applications. For example:

```
100488: DEBUG4:  root0.driver0.: Queued MemWr with request
    number 30 to address 0x2e980a7c
```

Requests initiated by the application are stored in a table. Each received completion is mapped to a request stored in the table with status of “ACTIVE” or “COMPLETE_PARTIAL”.

Search for “TransmitTLP” or ”ReceiveTLP” to see TLPs transmitted or received by the VIP.

Search for a transaction ID (for example XID = 0xd80014) to find related TLPs.

A transaction ID can be used to map a particular TLP to the transaction log (R_ID/Tag column) or to waveforms (ASCII in the Transaction Layer).

12.1.4 Logging Symbols

Symbol traffic for all VIP instances can be logged in a separate log, with symbols for each timestep and each instance on one line in the log. Use the phy layer parameter ENABLE_SYMBOL_LOG to enable and disable logging. To specify a file name other than the default "symbol_log", use the $msglog_control task with the MSGLOG_CONTROL_SET_SYMBOL_LOG_FILE command argument. See “Symbol Logging”.

12.2 Analyzing Runtime Statistics

Statistics about the current test are available during runtime. Separate statistics are logged for each layer and application.

The `DisplayStats()` function prints the internal statistics registers. You can also look in the Application_*app* directory and the TransactionLayer, LinkLayer, and PhyLayer directories for stat_num_*infokind*, to see what was logged for the statistic identified by stat_num_*infokind*.

12.3 Debugging Using the Protocol Analyzer

The Protocol Analyzer is a protocol-oriented analysis tool that can be used with the DVE tool to debug designs using Verification IPs (VIPs).

The DVE tool is a Graphical User Interface (GUI) that you can use for debugging SystemVerilog, VHDL, Verilog, and SystemC designs. You can drag-and-drop your signals in various views or use the menu options to view the signal source, trace drivers, compare waveforms, and view schematics.

The Protocol Analyzer tool supports the following PCIe features:

- TLPs
  - Request
  - One or more completions, as per protocol
- DLLPs
- Ordered Sets
- LTSSM state

12.3.1 Protocol Analyzer Settings File

Protocol Analyzer settings are saved in a file named ~/.pa/preferences.

You can reset screen settings to their defaults by deleting pa.pa.pg.

12.3.2 Enabling the Protocol Analyzer for PCIE

The Protocol Analyzer tool is off by default. You must enable it to get XML output for Protocol Analyzer.

Define PCIESVC_INCLUDE_PA:

```
vcs … +define+PCIESVC_INCLUDE_PA
```

Include the pciesvc_pa_pkg.sv file:

```
vcs … +define+PCIESVC_INCLUDE_PA \
  $EXPERTIO_PCIESVC_INSTALL_PATH/Include/pciesvc_pa_pkg.sv
```

12.3.3 Protocol Analyzer Controls

Enable the controls listed in Table 12-1 before link training is performed. These controls should be set only once because multiple on/off attempts may result in data being lost.

Table 12-1 Protocol Analyzer controls to set prior to link training

Control	Description
SetPALogFile (Transaction Layer)	Sets the log file name. Default: *instance_name*.pcie_svc.xml
SetPATransactionMode (Transaction Layer)	On/Off control of transaction logging. Default: Off.
SetPALinkMode (Link Layer)	On/Off control of DL logging of TLPs and DLLPs. Default: Off.
SetPAPhyMode (Physical Layer)	On/off control of Physical Layer logging of LTSSM state. Default: Off.

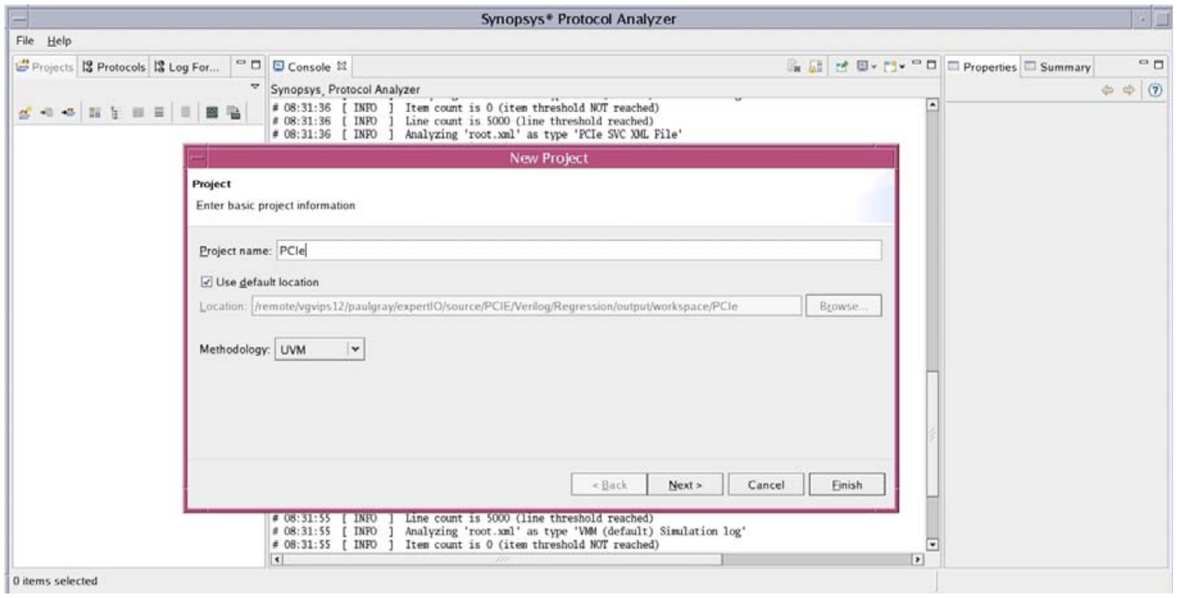
12.3.4 Protocol Analyzer Flow with DVE

1. Enable the Protocol Analyzer tool.
2. Compile the code

   Set up defines, include files, and enable debug (for waveform interaction with the DVE tool).
3. Create a Protocol Analyzer project. Figure 12-1 shows the dialog to create a project.
4. Import files. Add the .vpd and .tcl files to the project. The .tcl script builds the signal list in the waveform viewer.

   Figure 12-2 shows the Import Files dialog for adding files to the project.
5. Run the test.
6. Navigate with transaction and log views to examine the test results. Figure 12-3 shows the Protocol Analyzer transaction and log views combined in one GUI window.
7. Sync with the DVE tool.

Figure 12-1 Creating a Protocol Analyzer project



Synopsys, Protocol Analyzer

```
# 08:31:36  [ INFO  ]  Item count is 0 (item threshold NOT reached)
# 08:31:36  [ INFO  ]  Line count is 5000 (line threshold reached)
# 08:31:36  [ INFO  ]  Analyzing 'root.xml' as type 'PCIe SVC XML File'
```

Project

Enter basic project information

Project name: PCIe

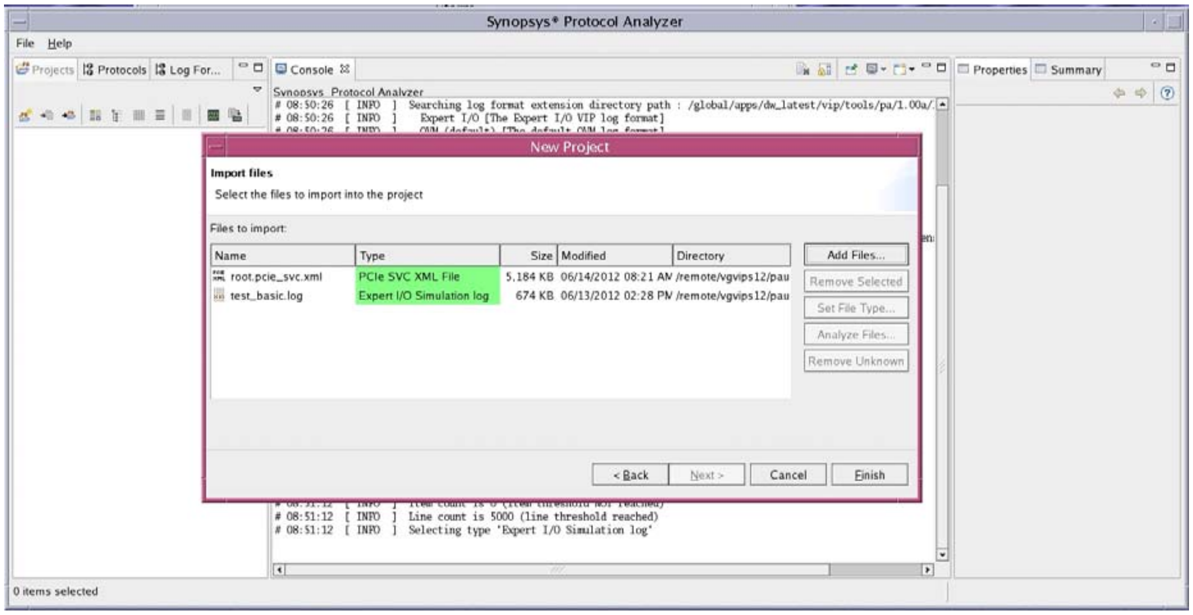
☑ Use default location

Location: /remote/vgvips12/paulgray/expertIO/source/PCIE/Verilog/Regression/output/workspace/PCIe Browse...

Methodology: UVM

```
# 08:31:55  [ INFO  ]  Line count is 5000 (line threshold reached)
# 08:31:55  [ INFO  ]  Analyzing 'root.xml' as type 'VMM (default) Simulation log'
# 08:31:55  [ INFO  ]  Item count is 0 (item threshold NOT reached)
```

Figure 12-2 Importing the .tcl and .vpd files to the project



Synopsys, Protocol Analyzer

```
# 08:50:26  [ INFO  ]  Searching log format extension directory path : /global/apps/dw_latest/vip/tools/pa/1.00a/
# 08:50:26  [ INFO  ]    Expert I/O [The Expert I/O VIP log format]
```

Import files

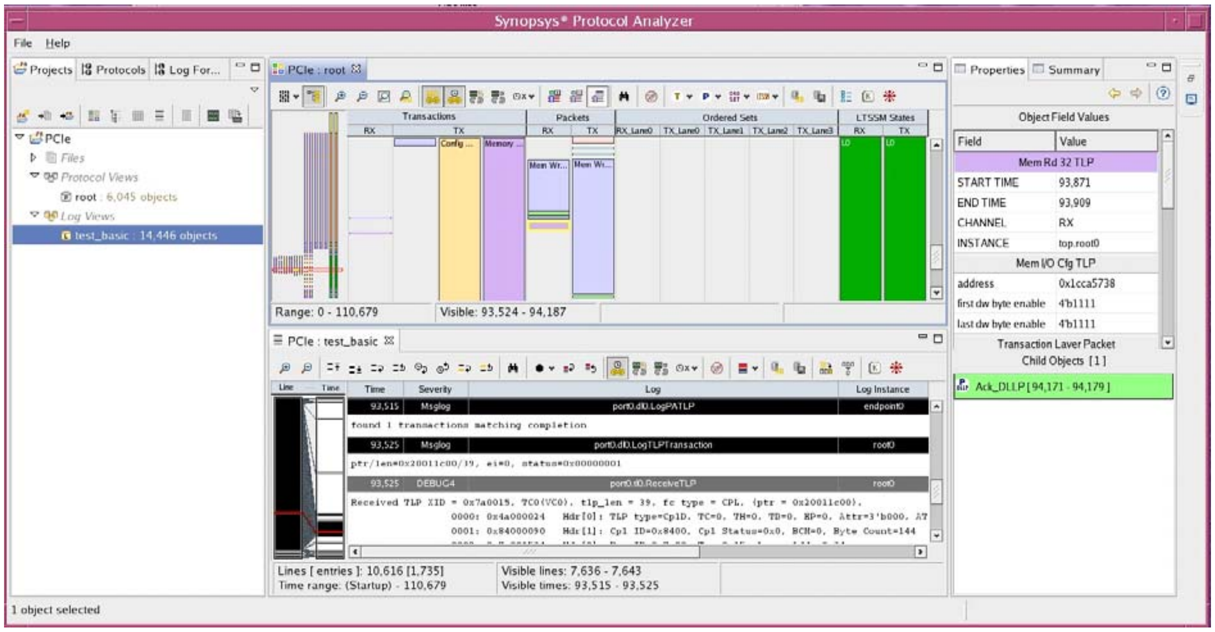
Select the files to import into the project

Files to import:

Name	Type	Size	Modified	Directory
root.pcie_svc.xml	PCIe SVC XML File	5,184 KB	06/14/2012 08:21 AM	/remote/vgvips12/pau
test_basic.log	Expert I/O Simulation log	674 KB	06/13/2012 02:28 PM	/remote/vgvips12/pau

```
# 08:51:12  [ INFO  ]  Line count is 5000 (line threshold reached)
# 08:51:12  [ INFO  ]  Selecting type 'Expert I/O Simulation log'
```

Figure 12-3 Protocol Analyzer transaction and log view



Range: 0 - 110,679
Visible: 93,524 - 94,187

Object Field Values

Field	Value
Mem Rd 32 TLP	
START TIME	93,871
END TIME	93,909
CHANNEL	RX
INSTANCE	top.root0
Mem I/O Cfg TLP	
address	0x1cca5738
first dw byte enable	4'b1111
last dw byte enable	4'b1111

Transaction Layer Packet
Child Objects [1]
Ack_DLLP [94,171 - 94,179]

```
 Time    Severity  Log                          Log Instance
 93,515  Msglog    port0.dl0.LogPATLP           endpoint0
found 1 transactions matching completion
 93,525  Msglog    port0.dl0.LogTLPTransaction  root0
ptr/len=0x20011c00/19, ei=0, status=0x00000001
 93,525  DEBUG4    port0.tl0.ReceiveTLP         root0
Received TLP XID = 0x7a0015, TC0(VC0), tlp_len = 39, fc type = CPL, (ptr = 0x20011c00),
              0000: 0x4a000024   Hdr[0]: TLP type=CplD, TC=0, TH=0, TD=0, EP=0, Attr=3'b000, AT
              0001: 0x84000090   Hdr[1]: Cpl ID=0x8400, Cpl Status=0x0, BCM=0, Byte Count=144
```

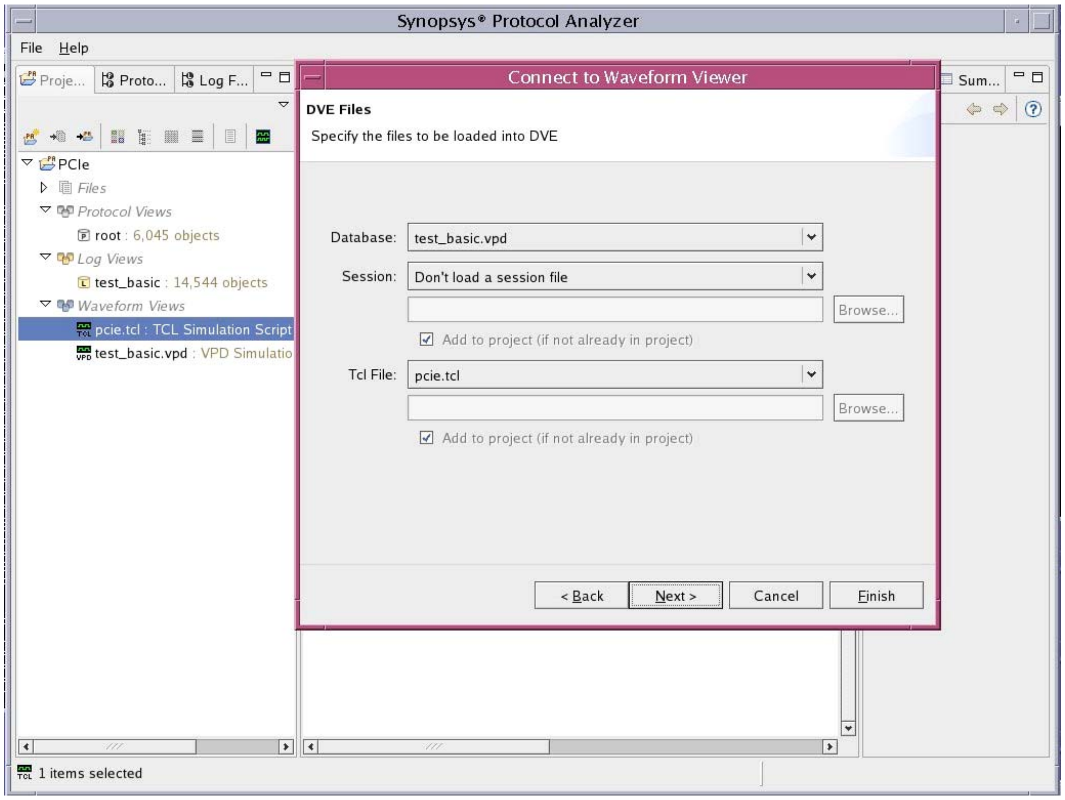
Lines [entries]: 10,616 [1,735]
Time range: (Startup) - 110,679
Visible lines: 7,636 - 7,643
Visible times: 93,515 - 93,525

12.3.5 Launching the DVE Connection Wizard

1. Select the .vpd or .tcl file under “Waveform Views”.
2. Click the DVE connection wizard button.
3. Make sure the option to run the tcl script is selected.
4. Verify that you have selected the default signal grouping.

Figure 12-4 shows the DVE Connection Wizard dialog for connecting to the DVE waveform viewer.

Figure 12-4 DVE connection wizard


Synopsys® Protocol Analyzer
File Help
Proje... Proto... Log F...
PCIe
Files
Protocol Views
root : 6,045 objects
Log Views
test_basic : 14,544 objects
Waveform Views
pcie.tcl : TCL Simulation Script
test_basic.vpd : VPD Simulatio
Sum...
1 items selected

Connect to Waveform Viewer

DVE Files

Specify the files to be loaded into DVE

Database: test_basic.vpd

Session: Don't load a session file

Add to project (if not already in project)

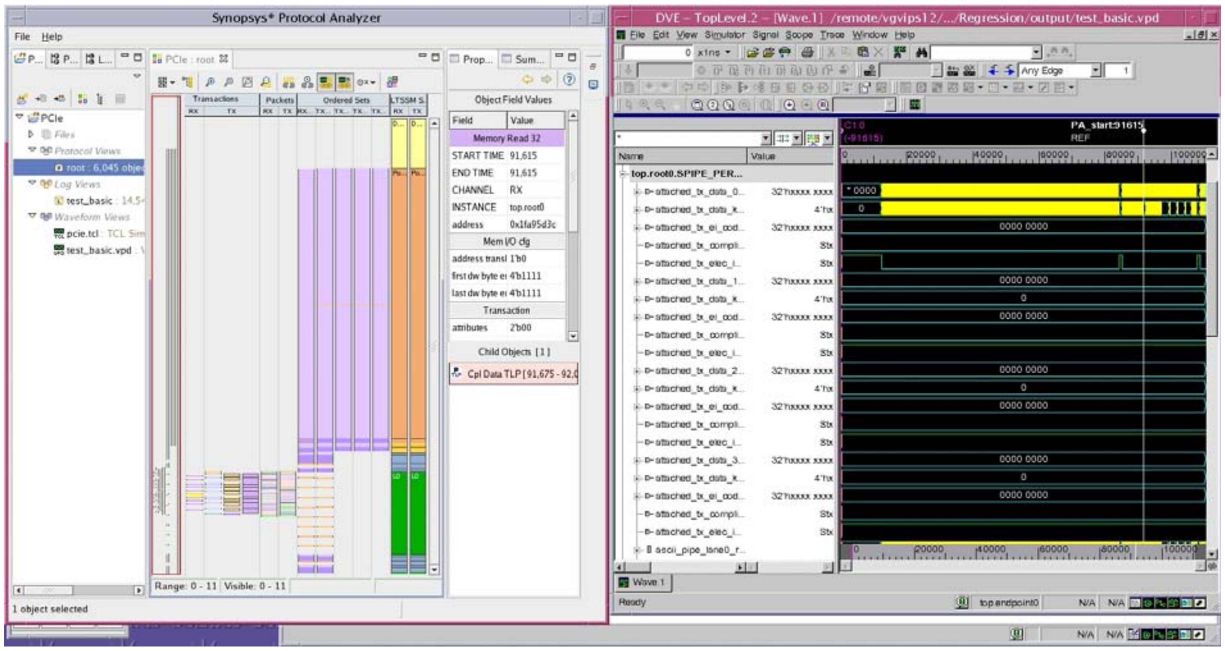
Tcl File: pcie.tcl

Add to project (if not already in project)

Browse...
Browse...
< Back
Next >
Cancel
Finish

Protocol Analyzer results and test output waveforms can then be viewed together to analyze and debug the test run. Figure 12-5 shows the Protocol Analyzer tool synced with the DVE tool.

Figure 12-5 Synchronized Protocol Analyzer and DVE tools



12.4 Using ASCII Signals

The following sections document the ASCII signals you can use for viewing within a waveform viewer. Note, you may see ASCII signals prefaced with “debug”. These are only for internal Synopsys use.

12.4.1 Transaction Layer ASCII Signals

The ASCII signals listed in Table 12-2 are available for viewing within a waveform viewer.   Access to the signals is through the XMR path to the TL. For example, in the examples shipped with the model, they are found at: “test_top.root0.port0.tl0.*ascii*”

Table 12-2 ASCII signals available for waveform viewers

Event Name	Description
ascii_rx_tlp_fc_type	Flow control credits associated with this TLP. Values are P, NP, CPL.
ascii_rx_tlp_type	Type of received TLP as defined by (fmt, type) fields.
ascii_rx_tlp_vc	VC on which TLP is received.
ascii_rx_tlp_xld	Received TLP transaction ID.
ascii_tx_tlp_fc_type	Flow control credits associated with this TLP. Values are P, NP, CPL.
ascii_tx_tlp_type	Type of TLP to be sent as defined by (fmt, type) fields.
ascii_tx_tlp_vc	VC on which TLP is sent.
ascii_tx_tlp_xld	Sent TLP transaction ID.

12.4.2 Data Link Layer ASCII Signals

ASCII signals on the Data Link Layer are listed in Table 12-3.

Table 12-3 Data Link Layer ASCII signals

Signal Name	Description
ascii_tx_tlp_type	Sent TLP type, defined by {fmt, type} fields.
ascii_tx_tlp_seq_num	Sent TLP sequence number.
ascii_tx_tlp_ei_code	TLP sent with this EI.
ascii_tx_dllp_type	Sent DLLP type.
ascii_tx_dllp_seq_num	Sent DLLP sequence number for ACK/NAK.
ascii_tx_dllp_credit_vc	Sent DLLP VC.
ascii_tx_dllp_credit_data_value	Sent DLLP data credit value.
ascii_tx_dllp_credit_hdr_value	Sent DLLP header credit value.
ascii_rx_tlp_type	Received TLP type, defined by {fmt, type} fields.
ascii_rx_tlp_seq_num	Received TLP sequence number.
ascii_rx_dllp_type	Received DLLP type.
ascii_rx_dllp_seq_num	Received DLLP sequence number for ACK/NAK.
ascii_rx_dllp_credit_vc	Received DLLP VC.
ascii_rx_dllp_credit_data_value	Received DLLP data credit value.
ascii_rx_dllp_credit_hdr_value	Received DLLP header credit value.
ascii_dlcmsm_state	Data Link Control Management State Machine.
ascii_vc[0-7]_fcsm_state	Flow Control State Machine for VC[0-7]
ascii_tx_dllp_ei_code	DLLP error codes.
ascii_aspm_state;	ASPM state
ascii_pm_state;	PM state
ascii_tx_callback;	Callback being executed on TX side.
ascii_rx_callback;	Callback being executed on the RX side.
ascii_fc_init_state;	Flow control init state

12.4.3 Physical Layer ASCII Signals

Physical Layer ASCII signals are listed in Table 12-4.

Table 12-4 Physical Layer ASCII signals

Signal Name	Description
ascii_ltssm_tx_state	LTSSM state of the transmitter
ascii_ltssm_rx_state	LTSSM state of the receiver
ascii_lane*n*_rx_data	Data received on lane n, where n is a number between 0 and 31.

Table 12-4 Physical Layer ASCII signals (Continued)

Signal Name	**Description**
ascii_lane*n*_tx_data	Data transmitted on lane n, where n is a number between 0 and 31.
ascii_pipe_lane*n*_rx_data;	Symbol data on received on PIPE lane *n*, where n is a number between 0 and 31.
ascii_pipe_lane*n*_tx_data	Symbol data on transmitted on PIPE lane *n*, where n is a number between 0 and 31.
ascii_hotplug_mode	Current state of hotplug mode
ascii_prev_hotplug_mode	Previous state of the hotplug.
ascii_lane_reversal_mode;	Lane reversal indicates if lane reversal is enabled.

13 Memory Model Utility

13.1 Overview

The Memory Model is a utility used throughout the VIP products offered by Synopsys. The memory model provides a mechanism to manage data, store data structures, and pass arguments in a Verilog environment. The combination of the VIP memory model and the design methodology employed by Synopsys provides many of the benefits of higher level, abstract test bench languages in a Verilog language environment.

The memory model (svc_mem.v) is instantiated in the top level of the user test bench. A product-specific Verilog define, `*XXX*SVC_MEM_PATH, where *XXX* is the product name, is assigned a value of that represents the Verilog hierarchy in which the model is instantiated. This define is understood by the VIP products offered by Synopsys when referencing utilities of the memory model.

13.2 Memory Model Interface

There are a suite of tasks that may be called to use the memory model. The tasks are described in Table 13-1.

Table 13-1 Memory Model interface tasks

Task Name	Arguments	I/O	Purpose	Description
AllocateMemory	display_string	I	The string displayed in the event of an error message (or DisplayMemStat when owner info is dumped). Typically an identifier of where the task is called.	This task allocates a contiguous block of memory. The size argument is in bytes and the mem_ptr is a 4 byte-aligned address. The mem_ptr returned will be page aligned. If the requested buffer cannot be reserved, the value -1 is returned and an error message output to the log file.
	size	I	Number of bytes to allocate	
	address	O	Byte address of the buffer that is allocated	

Table 13-1 Memory Model interface tasks (Continued)

Task Name	Arguments	I/O	Purpose	Description
FreeMemory	display_string	I	The string displayed in the event of an error message. Typically an identifier of where the task is called.	This task will free memory reserved starting at location mem_ptr for length of size bytes. The mem_ptr is a byte address that is 4 byte aligned. An error message will be displayed if the byte address is not 4-byte aligned, the address given has not previously been reserved, or the address given with a size of “0” is not recognized as start of page address.
	address	I	Byte address of the start of the buffer to free	
	size	I	Number of bytes to free. A value of “0” implies the entire memory page reserved pointed to by address	
WriteMemory	display_string	I	The string displayed in the event of an error message. Typically an identifier of where the task is called.	This task writes the data specified to the address specified. If the memory has not previously been allocated, an error will be written to the log file. The bytes written will start at the given address and continue for as many bytes supplied in the number of bytes input argument. The number of bytes may range from 1 to 4 bytes.
	address	I	Byte address to write data	
	data	I	32-bit data	
	num_of_bytes	I	Number of bytes to write starting at the given byte address	
ReadMemory	display_string	I	The string displayed in the event of an error message. Typically an identifier of where the task is called	Returns the 32 bit data starting at the byte address supplied. The memory must already be allocated. If not, an error message will be sent to the log file.
	address	I	Byte address for read data	
	data	O	32-bit data	
GetMemPageSize	mem_page_siz e	O	Size in bytes	The value of the page size that is being used for memory
IsMemoryEmpty	result	O	1 = Empty 0 = Not Empty	Parses allocated pages. If there is at least one page allocated, returns result of 0, not empty.

Table 13-1 Memory Model interface tasks (Continued)

Task Name	Arguments	I/O	Purpose	Description
DisplayMemStat	severity_level	I	The message log level used for the display	If the DISPLAY_MEM_STAT_OWNER_INFO parameter (see below) is set, this task will display ownership of the allocated pages, else it displays the current usage map of the 32-bit memory model. A 1 indicates a reserved page; a 0 is a free page. The severity level is used as an input to a $msglog() call.
NumPagesUsed	num_pages_used	O	Number of pages currently allocated.	This task returns the number of pages allocated in memory

13.3 Memory Size Parameters

There are several parameters used to control the size of the memory. The size of the memory instantiated will directly affect the memory footprint during a simulation. The default values provide a compromise between a reasonable size simulation footprint and offering enough memory for medium to large simulations. For simulations where more memory needed, Table 13-2 provides information about setting the appropriate parameters.

Table 13-2 Memory Model parameters

Parameter Name	Default	Description
DISPLAY_NAME	“svc_mem.”	The name that will be displayed in error and statistic message logging. Helps to distinguish the message from another memory model in the case of multiple instantiations.
MEM_SIZE_IN_PAGES	2048	The number of pages that in a single memory model. The memory footprint will be approximately the MEM_SIZE_IN_PAGES * PAGE_SIZE_IN_BYTES.
PAGE_SIZE_IN_BYTES	512	The number of bytes that constitute a page. When using in coordination with a Synopsys VIP, for best memory utilization, set the page size to the block size.
SMALL_MEM_SIZE_IN_PAGES	2048	The number of ‘small’ pages that in a single memory model. The memory footprint will be approximately the SMALL_MEM_SIZE_IN_PAGES * SMALL_PAGE_SIZE_IN_BYTES.
SMALL_PAGE_SIZE_IN_BYTES	32	The number of bytes that constitute a ‘small’ page . When using in coordination with a Synopsys VIP, for best memory utilization, the page size should be set to optimize memory use based on the required data structures.

Table 13-2 Memory Model parameters (Continued)

Parameter Name	Default	Description
DISPLAY_MEM_STAT_OWNER_INFO	0	If set to 1, the memory will keep track of who allocated the various memory blocks, and when the DisplayMemStat task is called, each owner is printed alongside the amount of memory allocated. If set to 0, the DisplayMemStat task will dump the block usage map.
DISPLAY_MEM_STAT_PERIOD	000_000	The time in nanoseconds that the memory model will display statistics to the log file. Statistics include such values as percentage of the memory used, and the current allocation bit map. If this value is zero, statistics will not be sent to the log file.

13.4 Model Instantiation

The memory model is typically instantiated in the top level of the test bench. The product specific define *XXX*SVC_MEM_PATH is then assigned a value that reflects the location in the hierarchy. See example code fragment below where the module test_top is the top-level verilog module of the test environment:

```
`SATASVC_MEM_PATH test_top.mem0
module test_top;
.
.
.
svc_mem mem0
.
.
.
endmodule
```

14 Message Logging Facility

14.1 Overview

The VIP message logging facility provides a mechanism for all components in the VIP to use a common logging facility. This provides not only a consistent interface, but also a single log for the outputs of the separate components.

In addition, the shared nature of the facility allows all modules in the simulation to implicitly make requests of each other to, for example request that another module suppress error message generation when the requestor is injecting a specific error.

14.1.1 Basic Logging Goals

When logging, there are several goals:

- Display adequate information to the user.
- Provide a means to count/show various errors that may occur in a standardized way.
- Allow the user to suppress error messages for cases that are purposely being injected.
- Avoid using excess resources (either CPU or disk).

To help meet these goals and allow for flexibility to the user, the facility provides for:

- A means to control the verbosity of displayed log events.
- A facility to remotely request that a particular log event's level be temporarily suppressed. It can be suppressed for a certain count, and after the expected stimulus has occurred, you can check if the stimulus response causing the message to be suppressed actually occurred (and how many times).
- Facilities to keep statistics on the logging events. Counters are available at each level of logging, as well as a means to stop the simulation at a particular count threshold.

14.1.2 Logging Levels

There are eight levels of logging events. These are divided into three levels of errors, four informative levels and one internal debug level. The logging levels are listed in Table 14-1.

Table 14-1 Logging event levels

Level	Definition
ERROR	Problems that will typically require the simulation to stop, i.e. FATAL – due to cascading errors or there is no means to recover.
WARNING	These are serious problems, but the simulation can continue at least for a while.
NOTICE	A detected problem that should be looked into, but not as serious as WARNING.
INFO	A general announcement of an informative nature that should normally be logged.
TRANSACTION	Notification of the start or end of a transaction in the VIP.
FSM / FRAME	Information about state-machine's state change or a full information about packet/frame/FIS to be sent or received.
DWORD	Details about individual primitives, data words and bytes on the interface.
DEBUG	Internal VIP debugging information. It can get quite verbose.

Note that levels ERROR, WARNING, and NOTICE are all considered to be errors and should be addressed prior to completion of verification.

14.2 Message Logging Utility Routines

Included are several utility routines to log and count messages. The advantages of using these routines over the standard `$display` task are:

- Message output format is standardized.
- Format controls are centralized – only the C file needs be changed, not the Verilog code.
- The number of messages for each logging level is counted and can be queried.
- Simulation can be automatically terminated when a maximum number of messages of a given level are exceeded.
- Message output can be remotely suppressed by another module to assist in writing directed and error-injecting tests.

The file named Verilog/Util/Msglog/msglog.c contains PLI tasks and functions for logging various levels of messages.

This should be included in a build by either dynamically linking a library that contains the compiled object file named msglog.o, or by explicitly static linking to msglog.o – depending on the particular Verilog simulator in use. See the Verilog/README file for additional information on dynamic and static PLI linking.

The following sections provide a reference for use of the various message logging routines.

14.2.1 PLI Task $msglog

Usage

`$msglog(log_level,[msg_code],` *`format`* `[,` *`args…`*`]);`

This task is used to display and log messages at a particular log level.

Task Name	Arguments	I/O	Description
$msglog	log_level (32)	I	The requested log level for the message
	msg_code (32)	I	Unique tag to mark this particular `$msglog` call for suppression purposes (optional argument).
	format (string)	I	The format of the message, akin to `$display`
	args (variable)	I	Optional arguments, akin to `$display`

The basic output format of the `$msglog` output is:

ttttt: *LOG_LEVEL: actual message*

Where *`ttttt`* is the current time tick, *`LOG_LEVEL`* is the human-readable log-level and the *`actual message`* is the format and args of the `$msglog` task call. An end-of-line character follows the output line.

Examples

```
$msglog(LOG_INFO, "The value of the data is %0d", 99);
```

This prints the following:

```
0: INFO: The value of the data is 99
```

Another example, with a message code tag:

```
$msglog(LOG_ERROR, MSG_CODE_XX_YY_SOME_ERR,
  "An error occurred – the actual/expected value was %0d/%0d",
  actual_data_value, expected_data_value)
```

Using the scope that the `$msglog` is coded in, and the above MSG_CODE_XX_YY_SOME_ERR tag, you can use the `$msglog_suppress()` call to request suppression of this error message. See "PLI Task $msglog_suppress" on page 14-245 for details about that task.

14.2.1.1 Message Logging Level Parameters

Below are the various log-levels (passed to `$msglog` and other tasks) are listed in Table 14-2. Although they are listed as parameters, using a defparam on their values with hard-coded values is not recommended. The numeric values of the parameters may change in future revisions of the code therefore the parameters should be used.

File name: .../Verilog/Util/svc_util_parms.v

Table 14-2 Message logging level parameters

Parameter Name	Description
LOG_ERROR LOG_ERR	Fatal logging level
LOG_WARNING LOG_WARN	Warning logging level
LOG_NOTICE LOG_NOTE	Notice logging level
LOG_INFO	Informational logging level

Table 14-2 Message logging level parameters (Continued)

Parameter Name	Description
LOG_TRANSACT	Transaction logging level
LOG_FSM	State-Machine and FIS/Frame logging level
LOG_FIS	
LOG_FRAME	
LOG_DWORD	Dword logging level
LOG_DEBUG	Debug logging level
LOG_DBG	
LOG_NUM_LEVELS	Number of logging levels.

Note that some of the log levels have different names for the same level (for example, LOG_ERROR and LOG_ERR). This is to allow multiple, often similar names to be used interchangeably, or to reflect that different protocols have different names for similar layers.

Note that while the various log level names (for example, LOG_INFO) and message code names (for example MSG_CODE_*) are generally fixed and should not change from release to release, their *values* are implementation specific. Avoid hard-coding values for these arguments.

14.2.1.2 Message Code Parameters

A message code is a 32-bit number used to uniquely tag a call to `$msglog`. This tag can be used to identify the specific `$msglog` call for use in message suppression. To organize these message codes, within each directory are typically files of the form *product_name*svc_*layer*_msgcodes.v" (for example, satasvc_transport_msgcodes.v). These files contain all the message codes for that layer.

A message code parameter is used as the optional second parameter of the `$msglog` PLI task call and has the general form:

MSGCODE_*PRODUCT-NAME*SVC_*LAYER_UNIQUE-NAME*

Example

```
MSGCODE_SATASVC_LINK_SLCC_OPEN_TIMEOUT_NOT_RECD
```

As with log levels, the various message codes should be identified by names not by their internal values, which might change.

14.2.2 PLI Task $msglog_clear

Usage

`$msglog_clear(`*`log_level`*`);`

This task is used to clear the counters of messages at the given *`log_level.`*

Task Name	Arguments (length)	I/O	Description
$msglog_clear	log_level (32)	I	The requested log level's counter to be cleared.

14.2.3 PLI Function $msglog_count

Usage

```
value = $msglog_count(log_level);
```

This function is used to get the current count of messages at *log_level* and below (for example, `$msglog_count(2)`returns the sum of all message counts of levels 0, 1 and 2.) Often this task is called at the end of a test run to display the count of any errors that may have occurred.

Task Name	Arguments (length)	I/O	Description
$msglog_count	log_level (32)	I	The requested log level's counter to be cleared.
	RETURNS: count (32)	O	The count of all messages logged at log_level and below.

Example

To count the number of message log events at log levels LOG_ERROR, LOG_WARNING and LOG_NOTICE:

```
count = $msglog_count(LOG_NOTICE);
```

To count the number of message log events only at log level LOG_WARNING:

```
count = $msglog_count(LOG_WARNING) - $msglog_count(LOG_ERROR);
```

14.2.4 PLI Task $msglog_exit_threshold

Usage:

```
$msglog_exit_threshold(log_level, count);
```

This task is used to set the count-threshold of `$msglog` displays at the given log level which will cause the simulation to exit.

Task Name	Arguments (length)	I/O	Description
$msglog_exit_threshold	log_level (32)	I	The log level associated with the given count.
	count (32)	I	The count (at ***log_level)*** that will cause the simulation to exit.

Usage example

To cause the simulation to exit after either 1 ERROR, 5 WARNINGS or 10 NOTICES,whichever comes first:

```
$msglog_exit_threshold(LOG_ERROR, 1);
$msglog_exit_threshold(LOG_WARNING, 5);
$msglog_exit_threshold(LOG_NOTICE, 10);
```

14.2.5 PLI Task $msglog_level

Usage:

```
$msglog_level(log_level [, prev_log_level]);
```

This task is used to set the log level at which messages are displayed. If the optional parameter `prev_log_level` is provided, it will be filled in with the previously set logging level. This controls how verbose the logging is.

Task Name	Arguments (length)	I/O	Description
$msglog_level	log_level (32)	I	The log level at (and below) which messages are displayed.
	prev_log_level (32)	O	The previous log level (optional task argument).

Example

```
integer prev_level;
  $msglog_level(LOG_NOTICE, prev_level);
    // set new log level, save previous
```

14.2.6 PLI Function $msglog_level_get

Usage

```
log_level = $msglog_level_get(0);
```

This task returns the current log level at which messages are being displayed.

Task Name	Arguments(length)	I/O	Description
$msglog_level_get	Unused but required	I	Function requires argument – typically pass 0. (Value is ignored.)
	RETURNS: count (32)	O	The current log level at (and below) which messages are displayed.

As with other tasks and functions dealing with message logging levels, ignore the level's value but compare it with existing manifest constants (for example, LOG_INFO) instead. Relative values, however, can be assumed. That is, a lower numerical value can be assumed to be a more critical logging level.

Example

```
integer cur_log_level;
iur_log_level = $msglog_level_get(0);
if (cur_log_level <= LOG_WARNING)
begin
   // something bad occurred... either LOG_ERROR or
   // LOG_WARNING
end
```

14.2.7 PLI Task $msglog_suppress

Usage

`$msglog_suppress(scope, msg_code, count, options);`

This task allows you to temporarily suppress the output of a specific call to `$msglog`. Often used for directed testing where an error is being injected, but the associated error message should not be displayed.

Task Name	Arguments (length)	I/O	Description
$msglog_suppress	scope (string)	I	String containing scope(s) to match, wildcards allowed.
	msg_code (32)	I	Specific $msglog call to suppress.
	count (32)	I	Number of times to suppress the $msglog call. Zero implies “forever”.
	options (32)	I	Reserved for future, set to 0.

The `$msglog` calls throughout the VIP contain the optional suppression tags if those calls are logging at level LOG_NOTICE, LOG_WARNING, or LOG_ERROR. The msglog codes are kept a parameter file within the director where the VIP source file is kept (so for example, if an VIP Link Layer task has been instrumented with suppression tags, those tags’ parameters will be listed in the file named …/LinkLayer/ xxxsvc_linklayer_msgcodes.v.) This file must be included in any file from which message suppression is being done.

The scope string must match exactly, but the following wildcards are allowed:

- * (asterisk) – matches any character, including no characters
- ? (question mark) – matches any single character

Be sure that the call to either `$msglog_unsuppress` or `$msglog_get_suppression_count` use the same scope string as was provided to the call to `$msglog_suppress`.

Example 14-1 shows how you could use the message suppression in a situation where there is a stimulus and a responder to that stimulus. The stimulus will be injecting an error to the responder. Rather than handling the expected error message, the stimulus will request that it be suppressed.

After the stimulus has been sent and it is expected that the responder has handled it, the message log suppression is then removed from the responder and the count is checked to verify that the expected suppression did occur.

Example 14-1

```
// This module is instantiated as top.host0.xport0.xx_yy0
module xx_yy(...);
   `include "Transport/satasvc_transport_msgcodes.v"
   // msg_codes here (example)
   task checker;
      begin
       ...
      if ... // some error condition
        begin
          ...
          $msglog(LOG_ERROR, MSG_CODE_XX_YY_SOME_ERR,
            "error message");
          ...
        end
      ...
    end
  endtask
endmodule

module stimulus(...);
  `include "Transport/satasvc_transport_msgcodes.v"
    // msg_codes here (example)
  task stim;
    integer options = 0;
    integer expected_err_count = 1;
    integer actual_err_count;
    begin
      ...
      // suppress the error
      $msglog_suppress("*.host0.xport0.xx_yy0",    //scope
        MSG_CODE_XX_YY_SOME_ERR,                //msg_code
        1, 0);                               //count, options

      // Request the stimulus that will cause error
      ...
      // After enough time/events have passed that the
      // error should have occurred, remove the
      // suppression:
      $msglog_unsuppress("*.host0.xport0.xx_yy0",
        MSG_CODE_XX_YY_SOME_ERR, actual_err_count);
      if (expected_err_count != actual_err_count)
        begin
          $msglog(LOG_NOTE, "...expected %0d errors,
            but actually got %0d", expected_err_count,
            actual_err_count);
        end
      ...
    end
  endtask
endmodule
```

The output of the above suppressed error message would look like:

```
12300: SUPPRESSED_0: error message
```

The error level has been replaced by SUPPRESSED_0 where the 0 indicates the level of the message that was being suppressed.

Note in the task `stim` in the above code fragment that the `$msglog_suppress` was called with a count of 1. This means that if the `$msglog` call that is being suppressed is called more than once, only the first one will be suppressed (and the actual suppression count that will be returned by `$msglog_unsuppress` will be a maximum value of 1.

14.2.8 PLI Task $msglog_unsuppress

Usage

```
$msglog_unsuppress(scope, msg_code, actual_count);
```

This task is used after the expected suppression(s) by `$msglog_suppress` have completed. It returns to the caller the number of suppressions that occurred.

Task Name	Arguments (length)	I/O	Description
$msglog_unsuppress	scope (string)	I	String containing scope(s) to match, wildcards allowed. This must match the scope string passed to $msglog_suppress *exactly*.
	msg_code (32)	I	Specific $msglog call to unsuppress.
	count (32)	O	Number of times the *$msglog* call was actually suppressed. When either the scope and/or msg_code do not match any existing suppression, it will be returned as -1.

See “PLI Task $msglog_suppress” on page 14-245 for an example.

14.2.9 PLI Task $msglog_get_suppression_count

Usage

```
$msglog_get_suppression_count(scope, msg_code, count);
```

This task is used to count the number of suppressions that have occurred for the given scope/code `$msglog` statement. This can be used to determine whether the suppressed error message has occurred yet and how many times.

Task Name	Arguments (length)	I/O	Description
$msglog_get_ suppression_count	scope (string)	I	String containing scope(s) to match, wildcards allowed.
	msg_code(32)	I	Specific *$msglog* call to count suppressions of.
	count (32)	O	Number of times the *$msglog* call was actually suppressed.

To get the current suppression count of a requested suppression:

```
integer count;
$msglog_get_suppression_count(
    "top.host0.xport0.xxx0",  // in
    MSGCODE_XXX0_SOME_SPECIFIC_CODE,  // in
    count  // out
);
if (count < 0)
  begin
    $msglog(LOG_NOTE, "Could not find suppression for
        scope=%0s/code=%0d",
        "top.host0.xport0.xxx0",         MSGCODE_XXX0_SOME_SPECIFIC_CODE);
  end
else $msglog(LOG_INFO, "The count for the given
    suppression was %0d", count);
```

14.2.10 PLI Task $msglog_control

Usage

`$msglog_control (request, parameter, args);`

This task is used for general purpose control of the Message Logging subsystem.

Task Name	Arguments (length)	I/O	Description
$msglog_control	command (32)	I	Command request to the Message Logging subsystem. See Table 14-3 for valid commands.
	parameter (32)	I	Detailed per-command parameters. See below for per-command parameters. Set to 0 if unused.
	args (32)	I	Additional arguments required by the specified command. See below.

Supported `$msglog_control` commands are listed in Table 14-3.

Table 14-3 msg_log control commands

	Command	Parameter (length)	Arguments
MSGLOG_CONTROL_SET_DISPLAY			
	Used to display debug information regarding the $msglog commands, in particular for helping to determine message codes and scope to use for message suppression.	0 (not used for this command)	MSGLOG_DISPLAY_SCOPE_ENABLED If set, show scopes only of suppressible msglog calls (those with a message code)
			MSGLOG_DISPLAY_SCOPE_SHOW_ SCOPE If set, show scopes only of suppressible msglog calls (those with a message code)

Table 14-3 msg_log control commands (Continued)

	Command	Parameter (length)	Arguments
			MSGLOG_DISPLAY_SCOPE_SHOW_ SUPPRESSABLE If this bit is set, show scopes only of suppressible $msglog calls (those with a message code).
			MSGLOG_DISPLAY_SCOPE_SHOW_ AT_START If this bit is set, show only dump of scopes at startup. (Only works at compile time.)
			MSGLOG_DISPLAY_MSG_CODE If this bit is set, show message codes of active $msglog calls.
MSGLOG_CONTROL_SET_LOG_COUNT_VARIABLE			
	Associate a variable in Verilog with a msglog counter – when $msglog updates its internal count for a given log-level, the associated variable is simultaneously updated.	log_level (32) This is the LOG_xxx log level that will be counted by the count variable.	Scope (string) Points to a string that contains the scope to an integer counter variable. Currently a full-scope is required (e.g. “top.err_count”)
MSGLOG_CONTROL_SET_LOG_FILE			
	Cause the simulation to write VIP logging to a user-specified log file.	0 (not used)	Filename (string) String containing the full or relative path to the log file. Default is simulator and/or command line dependent.

Table 14-3 msg_log control commands (Continued)

Command	Parameter (length)	Arguments
MSGLOG_CONTROL_SET_TRANSACTION_LOG_FILE: Cause the simulation to write VIP transaction logging to a user-specified log file. Note that this task is only useful for those protocols that support a transaction log.		Filename (string) This is the string that contains the full or relative path to the transaction log file or “” to disable the log. By default the transaction logger is disabled.
MSGLOG_CONTROL_SET_SYMBOL_LOG_FILE Set the file to be used for symbol logging.		Filename (string) This string contains the full or relative path to the symbol log file. Setting it to "" disables the log.

14.2.10.1 MSGLOG_CONTROL_SET_DISPLAY Example

This call will cause suppressed $msglog calls to show their called scope and message code. This is commonly used to setup appropriate calls to the $msglog_suppress task.

```
module top;
  initial
  begin
   $msglog_control(MSGLOG_CONTROL_SET_DISPLAY,
                   0,
                   (MSGLOG_DISPLAY_SCOPE_ENABLED
                    MSGLOG_DISPLAY_SCOPE_SHOW_SUPPRESSABLE
                    MSGLOG_DISPLAY_MSG_CODE)
                  );
  end
endmodule
```

14.2.10.2 MSGLOG_CONTROL_SET_LOG_COUNT_VARIABLE Example

To associate LOG_WARNING counter updates with a variable named my_warning_count:

```
module top;
  integer my_warning_count;
  initial
  begin
    $msglog_control(MSGLOG_CONTROL_SET_LOG_COUNT_VARIABLE,
                    LOG_WARN,    //Log level
                    “top.my_warning_count”);//Full scope
                                            //to variable
  end
endmodule
```

14.2.10.3 MSGLOG_CONTROL_SET_LOG_FILE Example

To redirect VIP logging to a new file named my_log_file in the “output” directory:

```
module top;
  reg [64*8-1:0] msglog_file;
  initial
    begin
      msglog_file = “output/my_log_file”;
      $msglog_control(MSGLOG_CONTROL_SET_LOG_FILE, 0,
         msglog_file);
  end
endmodule
```

14.2.10.4 MSGLOG_CONTROL_SET_TRANSACTION_LOG_FILE Example

To redirect VIP transaction logging to a new file named transaction_log in the /tmp directory after the simulation has run for 10,000 ticks, then shut it off after 500,000 more ticks have elapsed.

```
module top;
  reg [64*8-1:0] msglog_transaction_file;
               //Large enough string to hold filename
  initial
  begin
    #10_000;  //Do not start logging transactions until
              //10_000 ticks have gone by
    msglog_transaction_file = “/tmp/transaction_log”;
    $msglog_control(
        MSGLOG_CONTROL_SET_TRANSACTION_LOG_FILE,
        0,
        msglog_transaction_file
    );

    #500_000;  //Let the sim run for 500_000 more ticks
               //then shut of transaction logging
    msglog_transaction_file = "";  //Empty string will
                                   //close the transaction log
    $msglog_control(
      MSGLOG_CONTROL_SET_TRANSACTION_LOG_FILE,
      0,
      msglog_transaction_file
    );
  end
endmodule
```

14.2.10.5 MSGLOG_CONTROL_SET_SYMBOL_LOG_FILE Example

The following example shows how to

- Start symbol logging at time 0
- Specify a non-default symbol file name
- Stop symbol logging after 50,000 ticks
- Restart it after another 50,000 ticks.

```
module top;
...
```

```
reg [64*8-1:0] msglog_symbol_file;  //Large enough string to hold filename
defparam   top.<instance_path>.port0.pl0.ENABLE_SYMBOL_LOG = 1;

initial
  begin
    msglog_symbol_file = "/tmp/symbol_log";
    // Specify the log file name
    $msglog_control( MSGLOG_CONTROL_SET_SYMBOL_LOG_FILE, 0, msglog_symbol_file );
    #50_000;
    // Disable logging
    $msglog_control( MSGLOG_CONTROL_SET_SYMBOL_LOG_FILE, 0, "" );
    #50_000;
    // Re-enable logging
    $msglog_control( MSGLOG_CONTROL_SET_SYMBOL_LOG_FILE, 0, msglog_symbol_file );
  end
endmodule
```

A
Media Module

The SVC Media module is a Verilog module that provides a disk or tape (or other media) backing store. It efficiently uses memory to avoid the problems encountered with sparse storage situations, as are frequently encountered in a large disk drive.

The SVC Media Builder is a module that allows the user of the VC VIP for SATA to create disk image files that are pre-loadable to the media on a device or host. It is designed to be used as a standalone simulation to generate the media content file.

A medium is described by its ID number, type, and has a fixed block size. It consists of a set of one or more addressing fields that are used in the case of a disk medium to represent one or more LUNs and LBAs.

A.1 Media Module svc_media

The svc_media module provides a facility to hold the data for an initiator (host) or device. You first create the medium by calling `RegisterId`, or `LoadMedia` if you have a media content file. Once the medium is created, you use calls to the `WriteMedia` and `ReadMedia` tasks to put data to and get data from the medium.

A.1.1 Media ModuleTasks

Tasks used to access the instantiated media are described in Table A-1.

Table A-1 Tasks used to access instantiated media

Task Name	Arguments (length)	I/O	Argument Description	Task Description
RegisterId	media_id (64)	I	The media ID of this media	This task registers the particular media ID. Once a media is registered, one then uses the media ID as a 'handle' to access it via the Read and Write tasks.
	block_size (32)	I	The block size of this media.	
	num_luns (32)	I	Number of LUNs in this media.	
	media_type (8)		The media type of this media (for example, disk or tape). The file svc_media_parms.v contains the various MEDIA_TYPE_ values.	
	status (32)	O	Returned status for the result of this task call. The file svc_media_parms.v contains the various STATUS_ result parameter values.	
LoadMedia	filename (640)	I	The filename of the media content file.	This task requests the media load the given file. This will also implicitly call RegisterId with the parameters found in the media content file.
	status (32)	O	Returned status for the result of the LoadMedia. The svc_media_parms.v file contains the various STATUS_ result parameter values.	

Table A-1 Tasks used to access instantiated media (Continued)

Task Name	Arguments (length)	I/O	Argument Description	Task Description
ReadMedia	media_id (64)	I	The media ID of this media	This task reads the media and returns one data word. If it hasn't been registered, it will call RegisterId with the DEFAULT_BLOCK_SIZE, DEFAULT_NUM_LUNS and DEFAULT_MEDIA_TYPE.
	media_type(8)	I	The media type of this media	
	lun (64)	I	The LUN address. Not used (set to 0) when addressing a BUFFER media	
	lba (64)	I	The LBA to read from. Overload LBA with the unique BUFFER Id when accessing a BUFFER.	
	offset(32)	I	The byte offset to read from.	
	data (32)	O	The data word from the above ID/ LUN/LBA/OFFSET tuple.	
	found	O	Returns true if the data at the address is valid, else the data will be returned as 'X's.	
	status	O	Returned status for the result of this task call. The file svc_media_parms.v contains the various STATUS_ result parameter values.	

Table A-1 Tasks used to access instantiated media (Continued)

Task Name	Arguments (length)	I/O	Argument Description	Task Description
WriteMedia	media_id (64)	I	The media ID of this media	This task writes (at most) one data word to the media. If it hasn't been registered, it will call RegisterId with the DEFAULT_BLOCK_SIZE, DEFAULT_NUM_LUNS
	media_type(8)	I	The media type of this media	
	lun (64)	I	The LUN. When accessing a BUFFER media, this field is not used (set to 0).	
	lba (64)	I	The LBA. When addressing a BUFFER media, this field is overloaded with the buffer id.	
	offset(32)	I	The byte offset to write to.	
	data (32)	I	The data word from the above ID/ LUN/LBA/OFFSET tuple.	
	num_of_bytes (32)		The number of bytes in *data* that are valid.	
	found	O	Returns TRUE if this address being written-to has been previously written, else it will return FALSE if it is a write to a previously untouched section of the media.	
	status	O	Returned status for the result of this task call. The file svc_media_parms.v contains the various STATUS_ result parameter values.	
CountValidSectors	media_id (64)	I	The media ID of this media	Returns a count of the number of (contiguous) sectors in this ID/LUN/ LBA
	media_type(8)	I	The media type of this media.	
	lun (64)	I	The LUN to count.	
	lba (64)	I	The LBA to count.	
	count (32)	O	The returned sector count.	
InvalidateMedia	media_id (64)	I	The media ID to reference	Invalidate the LBAs of a given media. This task is used by the user to reset all LUNs / LBAs in the referenced media to an INVALID state. Typically used if the target DUT has been reset in the middle of the sim. The media will remain registers, but the content will be empty.
	media_type(8)	I	The media type of this media.	
	status	O	Returned status for the result of this task call. The file svc_media_parms.v contains the various STATUS_ result parameter values.	
DisplayInfo	severity_level (31)	I	Debug level to display the info.	Dump the contents of the media.

A.1.2 Media Module Parameters

The various parameters define the behavior of the instantiated media are listed in Table A-2. Note that there are two 'partitions' defined. The first partition (partition 0) is used to define block-based random access media such as disk drives. The second partition (partition 1) is typically used to define block-based sequential access storage media such as tape drives.

Note that the DEFAULT_BLOCK_SIZE can never be larger than the BLOCK_SIZE_LIMIT.

Table A-2 Parameters used to define instantiated media behavior

Parameter Name	Settable VAR?*	Default	Description
PARTITION0_MAX_NUM_UNIQUE_BLOCKS	No	2048	The number of pages in partition 0 of this media device.
PARTITION0_BLOCK_SIZE_LIMIT	No	528	The maximum number of bytes in a block for partition 0.
PARTITION1_MAX_NUM_UNIQUE_BLOCKS	No	0	The number of pages in partition 1 of this media device.
PARTITION1_BLOCK_SIZE_LIMIT	No	2048	The maximum number of bytes in a block for partition 1.
PARTITION2_MAX_NUM_UNIQUE_BLOCKS	No	4	The number of pages in partition 2 of this media device. This is primarily used for storing buffer data. A block represents a unique buffer id in this usage.
PARTITION2_BLOCK_SIZE_LIMIT	No	8192	The maximum number of bytes in a block for partition 2. This partition is primarily used to store buffer information (i.e., access through Read/Write Buffer SCSI commands). In the case of a buffer, the entire block is used as the buffer size.
INVALID_DATA_WORD	Yes	32'hx	Value returned when reading uninitialized data.
MAX_NUM_MEDIA_ID	No	128	The maximum number of total media IDs available for use.
DEFAULT_BLOCK_SIZE	No	512	Default block size to create when a media content file is not used to specify this.
DEFAULT_NUM_LUNS	No	32	Default number of LUNs to create per media when a media content file is not used to specify this.
DEFAULT_MEDIA_TYPE	No	MEDIA_TYPE_DISK The file svc_media_parms.v contains the various MEDIA_TYPE_ values.	Default media type to create when a media content file is not used to specify this.
LOAD_MEDIA_FILENAME_LIST_LEN	Yes	16 * 257	Space reserved for filename list – 16 filenames of 257 (256 + delimiter) bytes each.
LOAD_MEDIA_FILENAME_LIST	Yes	“”	List of DELIMETER separate filenames, total list length (including delimiters) must be < LOAD_MEDIA_FILENAME_LIST_LEN long.

Table A-2 Parameters used to define instantiated media behavior (Continued)

Parameter Name	Settable VAR?*	Default	Description
LOAD_MEDIA_FILENAME _DELIMETER	Yes	" " (space)	The separator between the filenames in the LOAD_MEDIA_FILENAME_LIST list. Default is a single space.
DISPLAY_NAME	No	"svc_media."	The name that will be displayed in error and statistic message logging. Helps to distinguish the message from another media model in the case of multiple instantiations.
*Settable VAR marked Yes implies the Parameter Name is also assigned to a Verilog signal in the module. The signal name is the same as the parameter name appended with "_VAR". This is useful if using higher level tools to set the parameters of the VIP. If marked No, only the parameter exists in the Verilog module.			

A.2 Media Builder (module: svc_media_builder)

This module is used standalone to create media files that are loadable by the VIP. Here are the steps of its use:

1. Instantiate the svc_media_builder module.
2. Set parameters (or _VARs) to control the various options of the media you want to create.
3. Call the `BuildMedia()` task to generate the file.

Note that these steps are separate from the simulation run for the VIP. The disk media file is created prior to running the VIP simulation

A.2.1 Media Builder Parameters

The parameters listed in Table A-3 are used to control the basic behavior of the SVC Media Builder. Note that most of the parameters also have a corresponding *`parameter`*`_VAR` version that is a Verilog reg, allowing it to be modified after instantiation. See "Media Builder Usage Example" for details.

Table A-3 Media Builder module parameters

Parameter Name	Type	Default	Description
MEDIA_ID	Integer	0	The unique Media ID for this media. Each created media needs to have a unique media ID.
MEDIA_TYPE	String	"DISK"	The particular Media Type for this media – valid types are "DISK", "TAPE", "CD" or "BUFFER"
BLOCK_SIZE	Integer	512	This media's block size.
LUN_LIST	String	"0"	List of LUNs created for this media. Comma separated.
MIN_LUN_SIZE	Integer	1	Minimum per-LUN length, in blocks.
MAX_LUN_SIZE	Integer	1	Maximum per-LUN length, in blocks.
STARTING_LBA_LIST	String	"0"	List of starting LBAs. Comma separated.
MIN_CONSEC_BLOCKS	Integer	1	Minimum consecutive blocks written after each starting LBA.

Table A-3 Media Builder module parameters (Continued)

Parameter Name	Type	Default	Description
MAX_CONSEC_BLOCKS	Integer	1	Maximum consecutive blocks written after each starting LBA. This value is constrained by the above STARTING_LBA_LIST; if the length would cause overrun from one LBA into another, the first will be truncated (with a message).
USE_INCR_DATA	Boolean	1	Media data generated will be incrementing words.
USE_RAND_DATA	Boolean	0	Media data generated is random words.
USE_MEDIA_ID_DATA	Boolean	0	Put Media ID in byte 3 of data (can be combined w/ USE_INCR_ or RAND_DATA)
USE_LBA_DATA	Boolean	0	Put LBA in byte 2 of data (can be combined w/ USE_INCR_ or RAND_DATA)
USE_LUN_DATA	Boolean	0	Put LUN in byte 1 of data (can be combined w/ USE_INCR_ or RAND_DATA)
USE_OFFSET_DATA	Boolean	0	Put Offset in byte 0 of data (can be combined w/ USE_INCR_ or RAND_DATA)
FILENAME	String	“svc_media_builder.out”	Filename to write the disk image.
DISPLAY_NAME	String	“svc_media.”	The name that will be displayed in error and statistic message logging. Helps to distinguish the message from another memory model in the case of multiple instantiations.

A.3 Media Initialization Content Data File

The data file to load the media consists of the parameters to describe the data loaded on the disk, as well as where it is actually loaded. The individual parameters are listed in Table A-4. The format is described below, followed by a usage example.

Table A-4 Media initialization data file parameters

Keyword Name	Parameter Values	Description
MediaID	64-bit numeric	The (unique) media identifier for this particular media. When a new media is defined, it must have a corresponding BlockSize (below). If not provided, defaults to previously set value.
MediaType	String	The specific type of the media. Must be provided immediately after ***MediaID***. Valid values are: “DISK”, “TAPE”, “BUFFER” and “CD”.
BlockSize	32-bit numeric	Sets the Block Size of the media identified by ***MediaID*** (in bytes). Must be provided immediately after ***MediaType***.
LUN	64-bit numeric	Logical Unit Number. Must be provided after ***BlockSize***. If not provided, defaults to previously set value.
LBA	64-bit numeric	Logical Block Address. Must be provided after **LUN**. If not provided, defaults to previously set value.

Table A-4 Media initialization data file parameters (Continued)

Keyword Name	Parameter Values	Description
DataOffset	0 - (BlockSize-1)	Data offset into LBA in bytes. Causes data to be written at this offset. Must be provided after **LBA**. If not provided, defaults to 0.
DataLength	32-bit numeric	Length of data to load, in bytes. Must be provided after the **LBA**.
Data	Space separated list of 32-bit values. Considered decimal unless prefixed with “0x”.	This is the actual data to be written to the media. Each value is a 32-bit dword; the number of these must match DataLength.

A.3.1 Media Initialization Data File Format

The media initialization data file consists of parameters that describe both the location of the data, as well as the actual data.

Each parameter is of the form "keyword = value". Any spaces between the keyword and value are ignored. The keyword and the value can be on the same line or on different lines. Mulltiple “keyword=value” pairs may be entered on one line, with one or more spaces between the “keyword=value” pairs.

All the keywords are strings (for example, "MEDIAID"). Case is ignored, so MediaID and mediaid are treated identically.

The particular parameters are somewhat hierarchical in that there are some that are dependent on others (for example, you must first provide the DataOffset and DataLength before providing the Data.) Not all the parameters are required in all cases: When practical, they will default from the previous values specified For example, when writing multiple LBAs, the LUN will default to the previously set value.

Verilog-style comments are allowed in the data file. They can be of block style, delimited by /* and */, or line style:

```
/*
* This is a block comment, and is used across
* multiple lines.
*/
// This line-style comment extends to the end of the line
```

Comments are not allowed between keywords and their values.

Values are interpreted as 32 or 64-bit wide decimal (0, 1, 2...) or, if a leading 0x is provided, as hexadecimal (0xabcd1234).

A.3.2 Media Builder Usage Example

The example below shows how to use the svc_media_builder module to create a disk image.

This example demonstrates the two ways to setup the media builder parameters, using `defparam` to set parameters at compile time, and using the _VAR variables to set the disk parameters at runtime. For each method, a call to `builder0.BuildMedia()` creates the file.

```
module build_media; // user defined module
   svc_media_builder builder0();
   // instantiate the media builder
   // By default we create one LUN
   // Create this LUN of random length between 1-200 blocks
```

```
      defparam builder0.MIN_LUN_SIZE = 1;
      defparam builder0.MAX_LUN_SIZE = 200;
      // for this LUN, start data creation at the following LBAs
      defparam builder0.STARTING_LBA_LIST =
         "10,20,30,400,500,600,700,800,900";
      // Each LBA will have a random length 1-100 blocks (constrained by
      // start list)
      defparam builder0.MIN_CONSEC_BLOCKS = 1;
      defparam builder0.MAX_CONSEC_BLOCKS = 100;
      // Write disk data file out to this filename:
      defparam builder0.FILENAME  = "test_media_disk1.out";
      initial
      begin
         #1; // wait for any initialization
         // Call Build Disk to build an image
         builder0.BuildMedia;
         // Now build another disk with ID=2 – note use of _VAR
         // variables – we can modify these after instantiation, unlike
         // the defparam statement:
         builder0.FILENAME_VAR = "test_media_disk2.out";
         builder0.MEDIA_ID_VAR = 2;
         // Create 3 LUNs:
         builder0.LUN_LIST_VAR = "0,1,2";
         // of length [10-100] blocks each…
         builder0.MIN_LUN_SIZE_VAR = 10;
         builder0.MAX_LUN_SIZE_VAR = 100;
         // Our list of starting LBAs, each of length [1-100] blocks
         builder0.STARTING_LBA_LIST_VAR = "1,2,4,8,16,32,64,128,256";
         builder0.MIN_CONSEC_BLOCKS_VAR = 1;
         builder0.MAX_CONSEC_BLOCKS_VAR = 100;
         // build it:
         builder0.BuildMedia;
      end
   endmodule
```

A.3.2.1 Sample Content File

Below is a sample media data file that demonstrates usage of all the keywords and constructs in the example above.

```
/* Sample Media Data File */
/* Block Comment. You can put it before or after parameter=value
   sections, but do nottry to do something like
   "parameter =  'block comment'  1234".
*/
MediaID=0x42    // Unique identifier
MediaType=Disk  // This media is a ‘disk’
BlockSize=512   // This is fixed for a given Media ID
LUNCount=1      // Number of LUNs to be defined
LUN= 1          // The first LUN
LBA = 0
DataOffset = 0
DataLength = 512
//
// Note that the first line of data below is decimal,
// the rest are hex
//
Data=
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0x42010010 0x42010011 0x42010012 0x42010013 0x42010014 0x42010015 0x42010016 0x42010017
0x42010018 0x42010019 0x4201001a 0x4201001b 0x4201001c 0x4201001d 0x4201001e 0x4201001f
0x42010020 0x42010021 0x42010022 0x42010023 0x42010024 0x42010025 0x42010026 0x42010027
0x42010028 0x42010029 0x4201002a 0x4201002b 0x4201002c 0x4201002d 0x4201002e 0x4201002f
0x42010030 0x42010031 0x42010032 0x42010033 0x42010034 0x42010035 0x42010036 0x42010037
0x42010038 0x42010039 0x4201003a 0x4201003b 0x4201003c 0x4201003d 0x4201003e 0x4201003f
0x42010040 0x42010041 0x42010042 0x42010043 0x42010044 0x42010045 0x42010046 0x42010047
0x42010048 0x42010049 0x4201004a 0x4201004b 0x4201004c 0x4201004d 0x4201004e 0x4201004f
0x42010050 0x42010051 0x42010052 0x42010053 0x42010054 0x42010055 0x42010056 0x42010057
0x42010058 0x42010059 0x4201005a 0x4201005b 0x4201005c 0x4201005d 0x4201005e 0x4201005f
0x42010060 0x42010061 0x42010062 0x42010063 0x42010064 0x42010065 0x42010066 0x42010067
0x42010068 0x42010069 0x4201006a 0x4201006b 0x4201006c 0x4201006d 0x4201006e 0x4201006f
0x42010070 0x42010071 0x42010072 0x42010073 0x42010074 0x42010075 0x42010076 0x42010077
0x42010078 0x42010079 0x4201007a 0x4201007b 0x4201007c 0x4201007d 0x4201007e 0x4201007f
LUN= 1
LBA = 1

// First word
DataOffset = 0
DataLength = 1
Data= 0x0       // Least significant byte in dword
DataOffset = 1  // NOTE: We do not enforce offset+length is on a
                // block boundary.
DataLength = 1
Data= 0x1
DataOffset = 2
DataLength = 1
Data= 0x1
DataOffset = 3
DataLength = 1
Data= 0x42    // Most significant byte in dword

// Second word
DataOffset = 4
DataLength = 2
Data= 0x0101

// Remainder of second, and start of third word - note implicit data
// offset
DataLength = 3
Data= 0x024201
DataLength = 3
Data = 0x420101
DataLength = 492
Data=
0x42010103 0x42010104 0x42010105 0x42010106 0x42010107 0x42010108 0x42010109 0x4201010a
0x4201010b 0x4201010c 0x4201010d 0x4201010e 0x4201010f
0x42010110 0x42010111 0x42010112 0x42010113 0x42010114 0x42010115 0x42010116 0x42010117
0x42010118 0x42010119 0x4201011a 0x4201011b 0x4201011c 0x4201011d 0x4201011e 0x4201011f
```

```
0x42010120 0x42010121 0x42010122 0x42010123 0x42010124 0x42010125 0x42010126 0x42010127
0x42010128 0x42010129 0x4201012a 0x4201012b 0x4201012c 0x4201012d 0x4201012e 0x4201012f
0x42010130 0x42010131 0x42010132 0x42010133 0x42010134 0x42010135 0x42010136 0x42010137
0x42010138 0x42010139 0x4201013a 0x4201013b 0x4201013c 0x4201013d 0x4201013e 0x4201013f
0x42010140 0x42010141 0x42010142 0x42010143 0x42010144 0x42010145 0x42010146 0x42010147
0x42010148 0x42010149 0x4201014a 0x4201014b 0x4201014c 0x4201014d 0x4201014e 0x4201014f
0x42010150 0x42010151 0x42010152 0x42010153 0x42010154 0x42010155 0x42010156 0x42010157
0x42010158 0x42010159 0x4201015a 0x4201015b 0x4201015c 0x4201015d 0x4201015e 0x4201015f
0x42010160 0x42010161 0x42010162 0x42010163 0x42010164 0x42010165 0x42010166 0x42010167
0x42010168 0x42010169 0x4201016a 0x4201016b 0x4201016c 0x4201016d 0x4201016e 0x4201016f
0x42010170 0x42010171 0x42010172 0x42010173 0x42010174 0x42010175 0x42010176 0x42010177
0x42010178 0x42010179 0x4201017a 0x4201017b 0x4201017c 0x4201017d
DataLength = 8
Data = 0x4201017e 0x4201017f
// Comments: you can put them before or after parameter=value
// sections, but do not try to do something like:
//   "KEYWORD = // comment that will not work correctly
//   1234".
//
LUN= 1  // MediaID, MediaType and BlockSize are already set, no need
        // to specify
LBA = 2
DataOffset = 0
DataLength = 512
Data=
0x42010200 0x42010201 0x42010202 0x42010203 0x42010204 0x42010205 0x42010206 0x42010207
0x42010208 0x42010209 0x4201020a 0x4201020b 0x4201020c 0x4201020d 0x4201020e 0x4201020f
0x42010210 0x42010211 0x42010212 0x42010213 0x42010214 0x42010215 0x42010216 0x42010217
0x42010218 0x42010219 0x4201021a 0x4201021b 0x4201021c 0x4201021d 0x4201021e 0x4201021f
0x42010220 0x42010221 0x42010222 0x42010223 0x42010224 0x42010225 0x42010226 0x42010227
0x42010228 0x42010229 0x4201022a 0x4201022b 0x4201022c 0x4201022d 0x4201022e 0x4201022f
0x42010230 0x42010231 0x42010232 0x42010233 0x42010234 0x42010235 0x42010236 0x42010237
0x42010238 0x42010239 0x4201023a 0x4201023b 0x4201023c 0x4201023d 0x4201023e 0x4201023f
0x42010240 0x42010241 0x42010242 0x42010243 0x42010244 0x42010245 0x42010246 0x42010247
0x42010248 0x42010249 0x4201024a 0x4201024b 0x4201024c 0x4201024d 0x4201024e 0x4201024f
0x42010250 0x42010251 0x42010252 0x42010253 0x42010254 0x42010255 0x42010256 0x42010257
0x42010258 0x42010259 0x4201025a 0x4201025b 0x4201025c 0x4201025d 0x4201025e 0x4201025f
0x42010260 0x42010261 0x42010262 0x42010263 0x42010264 0x42010265 0x42010266 0x42010267
0x42010268 0x42010269 0x4201026a 0x4201026b 0x4201026c 0x4201026d 0x4201026e 0x4201026f
0x42010270 0x42010271 0x42010272 0x42010273 0x42010274 0x42010275 0x42010276 0x42010277
0x42010278 0x42010279 0x4201027a 0x4201027b 0x4201027c 0x4201027d 0x4201027e 0x4201027f
/* EOF */
```

B
Protocol Checks

A number of automatic protocol checks are built into the PCIe VIP, to test for compliance with the PCIe specification. The following tables list the protocol checks, with a short description each check and the version of the PCIe specification in which each check is specified.

- Phy Layer protocol checks are listed in Table B-1.
- Data Layer protocol checks are listed in Table B-2.
- Application Layer protocol checks are listed in Table B-3.

Table B-1 Phy Layer protocol compliance checks

Protocol Check Name and Description	**Spec Version**
MSGCODE_PCIESVC_PHY_EXCESS_LATENCY_DIFF RxDeskewLanes - Excessive latency differential between lanes.	All
MSGCODE_PCIESVC_PHY_MISMATCHED_DATA_RATES This lane's data rate identifier does not match previous lane's.	>=2.1
MSGCODE_PCIESVC_PHY_DETECT_QUIET_TIMEOUT Timeout in state LTSSM_DETECT_QUIET.	All
MSGCODE_PCIESVC_PHY_DETECT_ACTIVE_TIMEOUT Timeout in state LTSSM_DETECT_ACTIVE.	All
MSGCODE_PCIESVC_PHY_POLLING_ACTIVE_TIMEOUT Timeout in state LTSSM_POLLING_ACTIVE.	All
MSGCODE_PCIESVC_PHY_POLLING_CONFIGURATION_TIMEOUT Timeout in state LTSSM_POLLING_CONFIGURATION	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_LINKWIDTH_START_TIMEOUT	All

Table B-1 Phy Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
Timeout in state LTSSM_CONFIGURATION_LINKWIDTH_START	
MSGCODE_PCIESVC_PHY_CONFIGURATION_LINKWIDTH_ACCEPT_TIMEOUT Timeout in state LTSSM_CONFIGURATION_LINKWIDTH_ACCEPT	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_LANENUM_ACCEPT_TIMEOUT Timeout in state LTSSM_CONFIGURATION_LANENUM_ACCEPT	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_LANENUM_WAIT_TIMEOUT Timeout in state LTSSM_CONFIGURATION_LANENUM_WAIT	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_COMPLETE_TIMEOUT Timeout in state LTSSM_CONFIGURATION_COMPLETE	>=2.1
MSGCODE_PCIESVC_PHY_CONFIGURATION_IDLE_TIMEOUT Timeout in state LTSSM_CONFIGURATION_IDLE	<=2.1
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRLOCK_TIMEOUT Timeout in state LTSSM_RECOVERY_RCVRLOCK	All
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_TIMEOUT Timeout in state LTSSM_RECOVERY_RCVRCFG	All
MSGCODE_PCIESVC_PHY_RECOVERY_IDLE_TIMEOUT Timeout in state LTSSM_RECOVERY_IDLE	All
MSGCODE_PCIESVC_PHY_WRONG_LINK_WIDTH Negotiated link width does not match expected link width.	All
MSGCODE_PCIESVC_PHY_RESERVED_SPEED_BIT0 SetSupportedSpeeds task sets rate bits in Tx TS OS. rate[0] is reserved but may be set using this task. Msg issued to prevent unintended results.	All
MSGCODE_PCIESVC_PHY_MAX_RX_SKP_INTERVAL ReceivePhy: Exceeded maximum interval before receiving a skp ordered set.	All
MSGCODE_PCIESVC_PHY_DATA_OUTSIDE_PACKET Received data outside of packet instead of logical idle.	All
MSGCODE_PCIESVC_PHY_BAD_LINK_NUMBER_POLLING_ACTIVE LTSSM: Incoming training sets have link number set to non-PAD value.	All
MSGCODE_PCIESVC_PHY_BAD_LANE_NUMBER_POLLING_ACTIVE . LTSSM: Incoming training sets have lane number set to non-PAD value	All
MSGCODE_PCIESVC_PHY_ELASTIC_BUFFER_OVERFLOW Elastic buffer has overflowed.	All
MSGCODE_PCIESVC_PHY_ELASTIC_BUFFER_UNDERFLOW	All

Table B-1 Phy Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
Elastic buffer has underflowed	
MSGCODE_PCIESVC_PHY_BAD_LANE_SKEW RxDeskewLanes: Lane skew has exceeded the maximum allowed lane skew.	>=2.1
MSGCODE_PCIESVC_PHY_TOO_MANY_SKP_SETS “Received skip ordered sets on a lane, but expected to receive at fewer skip ordered sets.”	All
MSGCODE_PCIESVC_PHY_TOO_FEW_SKP_SETS “Received skip ordered sets on a lane, but expected to receive at more skip ordered sets.”	All
MSGCODE_PCIESVC_PHY_COMMA_IN_PACKET Received unexpected K28.5 character in middle of a packet.	All
MSGCODE_PCIESVC_PHY_CONTROL_CHAR_IN_PACKET Received unexpected control character in middle of a packet.	All
MSGCODE_PCIESVC_PHY_INFER_ELEC_IDLE_SKIP Electrical idle has been inferred on all lanes due to lack of SKP ordered sets.	>=2.1
MSGCODE_PCIESVC_PHY_UNEXPECTED_POLARITY_CHANGE Detected rx_polarity go high but SVC hasn’t inverted polarity.	All
MSGCODE_PCIESVC_PHY_SPEED_CHANGE_NOT_IN_P0P1 Detected speed change while in powerdown is in state other than P0 or P1.	All
MSGCODE_PCIESVC_PHY_UNEXPECTED_PHYSTATUS Detected phystatus unexpectedly asserted.	All
MSGCODE_PCIESVC_PHY_PHYSTATUS_TIMEOUT Timeout waiting for phystatus acknowledgement.	All
MSGCODE_PCIESVC_PHY_VALID_SIGNAL_IN_P1 Detected a valid signal when attached device is supposed to be in P1 state.	All
MSGCODE_PCIESVC_PHY_VALID_SIGNAL_DURING_SPEED_CHANGE Detected valid signal on one or more lanes during a rate change.	All
MSGCODE_PCIESVC_PHY_TXDETRX_NOT_IN_P1 Receiver detect requested when powerdown not in P1 state.	All
MSGCODE_PCIESVC_PHY_DISABLED_TIMEOUT_NO_EIOS Timeout in this state. No EIOS was detected.	All
MSGCODE_PCIESVC_PHY_TXDETECTRX_DEASSERTED_BEFORE_PHYSTATUS Detected txdetectrx deasserted before phy_status acknowledgement.	All
MSGCODE_PCIESVC_PHY_SKP_INSERTED_NOT_ON_COMMA	All

Table B-1 Phy Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
SKP inserted status received on pipe interface. Data was not K28.5 (Comma).	
MSGCODE_PCIESVC_PHY_SKP_DELETED_NOT_ON_COMMA SKP deleted status received on pipe interface. Data was not K28.5 (Comma).	All
MSGCODE_PCIESVC_PHY_NO_LOOPBACK_PATTERN_DETECTED Did not detect transmitted loopback pattern on this lane before exiting loopback.	All
MSGCODE_PCIESVC_PHY_POLLING_COMPLIANCE_BAD_COMPLIANCE_RECEIVE Lane does not support 2.5GT/s. All lanes must advertise support for 2.5GT/s in compliance.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_START_NO_EIEOS LTSSM: Exceeded EIEOS interval of 32 without receiving an EIEOS.	>=2.0
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_UNEXPECTED_TS2 LTSSM: Received unexpected TS2 training set.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_UNEXPECTED_EIOS LTSSM: Received unexpected EIOS.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_UNEXPECTED_FTS LTSSM: Received unexpected FTS.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_BAD_LINK_NUM LTSSM: Detected incoming TS1 sets with link number set to value other than what SVC is configured to.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_LANE_NUM_NOT_PAD LTSSM: Incoming TS1 sets have lane number set to non-PAD.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_BAD_LANENUM LTSSM: Incoming TS1 sets have lane number set to wrong value.	All
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRLOCK_NO_EIEOS LTSSM: Exceeded EIEOS interval of 32 without receiving an EIEOS in this state.	>=2.0
MSGCODE_PCIESVC_PHY_RECOVERY_SPEED_ELECTRICAL_IDLE_WITH_NO_EIOS LTSSM: Electrical idle detected on this lane without preceding EIOS.	All
MSGCODE_PCIESVC_PHY_RECOVERY_SPEED_INFERRED_ELECTRICAL_IDLE LTSSM: Inferred electrical idle on this lane.	All
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_NO_EIEOS LTSSM: Exceeded EIEOS interval of 32 without receiving an EIEOS.	>=2.0
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_INFERRED_ELECTRICAL_IDLE LTSSM: Inferred electrical idle on this lane.	All
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_BAD_LANE_SKEW	All

Table B-1 Phy Layer protocol compliance checks (Continued)

Protocol Check Name and Description	**Spec Version**
LTSSM: TIMEOUT in this state - Unable to deskew lanes.	
MSGCODE_PCIESVC_PHY_HOT_RESET_NO_TS1_ACKNOWLEDGEMENT LTSSM: Hot Reset timer expired without receiving ts1 hot reset handshake.	All
MSGCODE_PCIESVC_PHY_RX_N_FTS_TIMEOUT "N_FTS timeout, SKP OS not received after N_FTS ordered sets."	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_COMPLETE_MISMATCHED_N_FTS LTSSM: This lane's N_FTS value does not equal previous lane's.	>=2.1
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_MISMATCHED_N_FTS LTSSM: This lane's N_FTS value does not equal previous lane's.	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_COMPLETE_BAD_LANE_SKEW LTSSM: Unable to deskew lanes.	>=2.1
MSGCODE_PCIESVC_PHY_LOOPBACK_ACTIVE_MISSING_EIOS Detected electrical idle on this lane without first detecting EIOS.	All
MSGCODE_PCIESVC_PHY_L0S_FTS_TIMEOUT LTSSM: N_FTS timeout on receiver in L0s.	All
MSGCODE_PCIESVC_PHY_L0_SIGNAL_LOSS Electrical idle detected on all lanes without first detecting EIOS.	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_COMPLETE_SUPPORTED_SPEEDS_CHANGE Detected change of supported_speeds field. This field must not change in state.	All
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_SUPPORTED_SPEEDS_CHANGE Detected change of supported_speeds field. This field must not change in state.	>=2.1
MSGCODE_PCIESVC_PHY_BAD_SPEED_CHANGE_REQUEST Speed change was requested but SVC did not advertise support for speeds greater than 2.5GT/s.	All
MSGCODE_PCIESVC_PHY_LOOPBACK_ENTRY_NO_COMPLIANCE_TIMEOUT Timeout in this state. No TS1 was detected with loopback enable set.	All
MSGCODE_PCIESVC_PHY_L0S_BAD_RX_FTS_COUNT Lane's FTS transmit count did not equal expected count before exiting L0S.	All
MSGCODE_PCIESVC_PHY_VALID_SIGNAL_DURING_PO_POWERDOWN_CHANGE Detected valid signal on one or more lanes before SVC has acknowledged transition to P0.	All
MSGCODE_PCIESVC_PHY_WRONG_PIPE_CLK_FREQ Pipe clock period does not match expected period.	All
MSGCODE_PCIESVC_PHY_LANE_REVERSAL_DETECTED	All

Table B-1 Phy Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
SVC has detected a lane reversal. Swapping lane assignments.	
MSGCODE_PCIESVC_PHY_MISMATCHED_LINK_UPCONFIGURE_BIT This lane's link upconfigure bit does not match previous lane's.	>=2.1
MSGCODE_PCIESVC_PHY_CONFIG_NON_PAD_LINK_LANE_NUM_FOR_NON_LINK_LANE LINK/LANE not PAD for lanes >= this lane. This lane is no longer part of the link.	All
MSGCODE_PCIESVC_PHY_CONFIG_NON_PAD_LINK_LANE_NUM_FOR_REVERSED_NON_LINK_LANE LINK/LANE not PAD for reversed lanes <= this lane. This lane is no longer part of the link.	All
MSGCODE_PCIESVC_PHY_RX_START_BLOCK_GEN1_GEN2 Detected rx_start_block != 0 asserted at speed less than 8G.	>=3.0
MSGCODE_PCIESVC_PHY_RX_SYNC_HEADER_GEN1_GEN2 Detected rx_sync_header != 0 at speed less than 8G.	>=3.0
MSGCODE_PCIESVC_PHY_RX_DATAK_GEN3 Detected nonzero value on pipe signal rx_datak at 8G.	>=3.0
MSGCODE_PCIESVC_PHY_8G_RATE_PIPE2 Rate set to 8G with the pipe spec version set to 2. Please check your configuration.	>=3.0
MSGCODE_PCIESVC_PHY_BLOCK_ALIGN_CONTROL_GEN1_GEN2 Detected block_align_control asserted at speed other than 8G.	>=3.0
MSGCODE_PCIESVC_PHY_RX_STP_TOKEN_BAD_FCRC_GEN3 Detected bad FCRC on incoming STP token.	>=3.0
MSGCODE_PCIESVC_PHY_RX_STP_TOKEN_BAD_PARITY_GEN3 Detected bad parity on incoming STP token.	>=3.0
MSGCODE_PCIESVC_PHY_RX_SDP_TOKEN_BAD_GEN3 Unexpected data in SDP token.	>=3.0
MSGCODE_PCIESVC_PHY_RX_TOKEN_MISMATCH_GEN3 Received token does not map to a recognizable token.	>=3.0
MSGCODE_PCIESVC_PHY_RX_NON_IDL_TOKEN_GEN3 Received non-IDL token outside of packet.	>=3.0
MSGCODE_PCIESVC_PHY_DSP_TS2_EQ_SYM6_BIT7_MUST_BE_1_IN_RCVRCFG "rx EQ TS2 symbol6, bit 7 should be 1 for Downstream port in Recovery.RcvrCfg @ 2.5G/5G."	>=3.0
MSGCODE_PCIESVC_PHY_USP_TS2_SYM6_MUST_BE_45_NOT_EQ_IN_RCVRY rx EQ TS2 symbol6 should be x45 for Upstream port in Recovery @ 2.5G/5G.	>=3.0
MSGCODE_PCIESVC_PHY_RX_INVALID_BLOCK_SIZE_GEN3	>=3.0

Table B-1 Phy Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
CheckBlockSize: block size not 16.	
MSGCODE_PCIESVC_PHY_DUT_ASSERTED_CLKREQ_L1_2_ENTRY “LTSSM: CLKREQ# asserted by DUT. SVC is in L1_2_ENTRY, not L1_2_IDLE.”	>=3.0
MSGCODE_PCIESVC_PHY_EQUALIZATION_BYPASSED This lane bypassed Equalization. Received TS2 in this state @ 8G when equalization_complete == 0.	>=3.0
MSGCODE_PCIESVC_PHY_EQUALIZATION_PHASE_2_NOT_ENABLED This lane executed Equalization Phase despite setting of ENABLE_EQUALIZATION_PHASE_VAR.	>=3.0
MSGCODE_PCIESVC_PHY_L0S_TOO_MANY_FTS_BETWEEN_EIEOS Received more than 32 FTS without detecting an EIEOS on this lane.	>=3.0
MSGCODE_PCIESVC_PHY_L0S_TOO_FEW_FTS_BETWEEN_EIEOS Received less than 32 FTS without detecting an EIEOS on this lane.	>=3.0
MSGCODE_PCIESVC_PHY_UNEXPECTED_COEFFICIENTS_VALID PIPE signal get_local_preset_coefficients asserted unexpectedly.	>=3.0
MSGCODE_PCIESVC_PHY_COEFFICIENTS_VALID_ASSERTED_TOO_LONG PIPE signal local_tx_coefficients_valid asserted for more than 1 clk cycle.	>=3.0
MSGCODE_PCIESVC_PHY_RXEQEVAL_DEASSERTED_EARLY PIPE signal RxEqEval deasserted before the evaluation was signaled complete.	>=3.0
MSGCODE_PCIESVC_PHY_GET_LOCAL_PRESET_COEFFICIENTS_TIMEOUT Timeout for get local preset coeffecient request.	>=3.0
MSGCODE_PCIESVC_SERDES_NEW_MIN_SEEN New min half bit period seen - SERDES unlocked.	All
MSGCODE_PCIESVC_SERDES_CLK_SLOWDOWN “Violation of 5 same bits period seen or New larger bit seen, but not at least 2x old bit - clock has likely slowed down - SERDES unlocked.”	All
MSGCODE_PCIESVC_SERDES_BYTE_UNLOCK “Lost valid signal level on receiver, PLL (clock) recovery reset.”	All
MSGCODE_PCIESVC_SERDES_PLL_UNLOCK “Lost valid signal level on receiver, PLL (clock) recovery reset.”	All
MSGCODE_PCIESVC_SERDES_LOSS_OF_BYTE_SYNC “Comma character detected in other than byte 0, bit 0. SERDES unlocked.”	All
MSGCODE_PCIESVC_ENDEC_ILLEGAL_ENCODING “Encoder: Kcode requested, but no legal encoding found!”	All
MSGCODE_PCIESVC_ENDEC_ILLEGAL_DECODE	All

Table B-1 Phy Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
Decoder: Illegal decode received!	
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_N_FTS Detected illegal control character in N_FTS field of TS1.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_POLARITY_INVERSION Lane polarity inversion detected on incoming TS1 ordered set.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_OVERSIZE_EIOS Detected oversize EIOS.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_UNDERSIZE_EIOS Detected undersize EIOS.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_OVERSIZE_FTS Detected oversize FTS.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_UNDERSIZE_FTS Detected undersize FTS.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_OVERSIZE_SKIP Detected oversize SKP.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_UNDERSIZE_SKIP Detected undersize SKP.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_LINKNUM Detected illegal control character in Linknum field of TS1.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_LANENUM Detected illegal control character in Lanenum field of TS1.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_DATA_RATE Detected illegal control character in Data Rate Identifier field of TS1.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_TRAINING_CTRL Detected illegal control character in Training Control field of TS1.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_LINKNUM Detected illegal control character in Linknum field of TS2.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_LANENUM Detected illegal control character in Lanenum field of TS2.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_N_FTS Detected illegal control character in N_FTS field of TS2.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_DATA_RATE	All

Table B-1 Phy Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
Detected illegal control character in Data Rate Identifier field of TS2.	
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_TRAINING_CTRL Detected illegal control character in Training Control field of TS2.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_UNEXPECTED_COMMA Received K28.5 (Comma) in ordered set.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_ORDERED_SET Received bad ordered set.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_SKP_INSERT_STATUS Received SKP insert status on non-SKP ordered set.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_LINK_PAD_LANE_NON_PAD Received TS1 with PAD link number and non-PAD lane number.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_LINK_PAD_LANE_NON_PAD Received TS2 with PAD link number and non-PAD lane number.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_SKIP_SIZE_8G “Invalid SKP ordered set length. Valid lengths are 8,12,16,20 and 24.”	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS1_SYMBOL_14_15_DC_BALANCE_REDUCE_ONES “Expected TS1 symbols 14 and 15 = {8’h20, 8’h08} for dc balance.”	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS1_SYMBOL_14_15_DC_BALANCE_REDUCE_ZEROES “Expected TS1 symbols 14 and 15 = {8’hdf, 8’hf7} for dc balance.”	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS1_SYMBOL_15_DC_BALANCE_REDUCE_ONES Expected TS1 symbol 15 = 8’h08 for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS1_SYMBOL_15_DC_BALANCE_REDUCE_ZEROES Expected TS1 symbol 15 = 8’hf7 for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_SYMBOL_15_DC_BALANCE_BYTE14 Expected TS1 symbol 14 = 8’h4a for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_IDENTIFIER_BYTE14 Bad TS1 identifier on byte 14.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_IDENTIFIER_BYTE15 Bad TS1 identifier on byte 15.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS2_SYMBOL_14_15_DC_BALANCE_REDUCE_ONES “Expected TS2 symbols 14 and 15 = {8’h20, 8’h08} for dc balance.”	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS2_SYMBOL_14_15_DC_BALANCE_REDUCE_ZEROES	>=3.0

Table B-1 Phy Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
"Expected TS2 symbols 14 and 15 = {8'hdf, 8'hf7} for dc balance."	
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS2_SYMBOL_15_DC_BALANCE_REDUCE_ONES Expected TS2 symbol 15 = 8'h08 for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS2_SYMBOL_15_DC_BALANCE_REDUCE_ZEROES Expected TS2 symbol 15 = 8'hf7 for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_SYMBOL_15_DC_BALANCE_BYTE14 Expected TS2 symbol 14 = 8'h45 for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_IDENTIFIER_BYTE14 Bad TS2 identifier on byte 14.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_IDENTIFIER_BYTE15 Bad TS2 identifier on byte 15.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_INVALID_BLOCK Received invalid ordered set block.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_EIOS_5G_2_NOT_RECEIVED SVC Did not receive 2 back to back EIOS from DUT at 5G. Only recieved 1 EIOS.	>=2.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_SKP_8G_SCRAMBLER_VALUE The LFSR value contained in the SKP ordered set does not match the receive scrambler value.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_SKP_8G_PARITY SKIP ordered set parity does not match expected value.	>=3.0
MSGCODE_PCIESVC_PCS_INVALID_SYNC_HEADER ReceivePMA: Invalid sync hdr detected.	>=3.0
MSGCODE_PCIESVC_PCS_SKP_END_NOT_DETECTED SKP_END was not detected before byte 20.	>=3.0
MSGCODE_PCIESVC_PHY_RX_SKP_NOT_ALL_LANES_8G Detected 8G SKP on at least 1 lane but not all.	>=3.0
MSGCODE_PCIESVC_PHY_RX_SKP_LEN_ALL_LANES_8G Detected 8G SKPs with different lengths.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_SKP_MISSING_END_8G "Received SKP did not have SKP End indication on byte 4, 8, 12, 16 or 20."	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_SKP_8G Received start of next training set before previous SKP has completed.	>=3.0

Table B-2 Data Layer protocol compliance checks

Protocol Check Name and Description	Spec Version
MSGCODE_PCIESVC_DL_RECEIVE_TLP_LCRC ProcessReceivedTLP: Received TLP with bad LCRC	All
MSGCODE_PCIESVC_DL_RECEIVE_ILLEGAL_SEQ_NUM ProcessReceivedTLP: Received TLP with invalid sequence number.	All
MSGCODE_PCIESVC_DL_RECEIVE_NULLIFIED_TLP ProcessReceivedTLP: Received nullified TLP.	All
MSGCODE_PCIESVC_DL_RECEIVE_NULLIFIED_TLP_LCRC ProcessReceivedTLP: Received TLP with EDB delimiter and non inverted or bad LCRC	All
MSGCODE_PCIESVC_DL_RECEIVE_DUPLICATE_SEQ_NUM ProcessReceivedTLP: Received duplicate TLP.	All
MSGCODE_PCIESVC_DL_RECEIVE_DLLP_CRC ProcessReceivedDLLP: Received DLLP crc error.	All
MSGCODE_PCIESVC_DL_REPLAY_TIMER_TIMEOUT UpdateReplayTimer: Replay timer timed out.	All
MSGCODE_PCIESVC_DL_REPLAY_LATE_TIMEOUT CheckAttachedTimers: Replay received late.	All
MSGCODE_PCIESVC_DL_REPLAY_EARLY_TIMEOUT "ProcessReceivedDLLP: Retry rcvd too early, attached replay timeout "	All
MSGCODE_PCIESVC_DL_RECEIVED_ACK_EXP_REPLAY_TIMEOUT "ProcessReceivedDLLP: EI - Rcvd ACK, replay timeout was expected."	All
MSGCODE_PCIESVC_DL_RECEIVED_NAK_EXP_REPLAY_TIMEOUT ProcessReceivedDLLP: EI - Received NAK when a replay timeout was expected.	All
MSGCODE_PCIESVC_DL_RETRY_BUFFER_FULL_BYTES GetNextTLP: Retry buffer full. consumed bytes > MAX_NUM_RETRY_BUFFER_DWORDS.	All
MSGCODE_PCIESVC_DL_RETRY_BUFFER_FULL_PKTS GetNextTLP: Retry buffer full. Num Pkts exceeded MAX_NUM_RETRY_BUFFER_PKTS.	All
MSGCODE_PCIESVC_DL_RECEIVED_DUPLICATE_ACK ProcessReceivedDLLP: Received duplicate ACK.	All
MSGCODE_PCIESVC_DL_RECEIVED_UNEXPECTED_NAK ProcessReceivedDLLP: Received Unexpected NAK.	All
MSGCODE_PCIESVC_DL_RECEIVED_UNEXPECTED_NAK_ACK ProcessReceivedDLLP: Received unexpected NAK which ACK'd some packets.	All
MSGCODE_PCIESVC_DL_RECEIVED_NAK_DURING_REPLAY	All

Table B-2 Data Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
"ProcessReceivedDLLP: EI - Received NAK as expected, but replay already in progress."	
MSGCODE_PCIESVC_DL_RECEIVED_MULT_NAK_DURING_REPLAY "ProcessReceivedDLLP: Received more NAK during replay, expected 1. "	All
MSGCODE_PCIESVC_DL_RECEIVED_ACK_EXP_NAK ProcessReceivedDLLP: Received ACK when NAK expected.	All
MSGCODE_PCIESVC_DL_RECEIVED_ACK_PROTOCOL_VIOL "ProcessReceivedDLLP: Rcvd ACK with wrong sequence number, protocol violation ((((next_transmit_seq - 1) - acknak_seq_num) % 4096) > 2048)."	All
MSGCODE_PCIESVC_DL_RECEIVED_ACK_SEQ_NUM_NOT_FOUND PurgeRetryBuffer: Retry buffer is empty. seq num not found in retry buffer.	All
MSGCODE_PCIESVC_DL_RECEIVED_NAK_PROTOCOL_VIOL ProcessReceivedDLLP: Received NAK with protocol violation ((((next_transmit_seq - 1) - acknak_seq_num) % 4096) > 2048).	All
MSGCODE_PCIESVC_DL_RECEIVED_NAK_EXP_ACK PurgeRetryBuffer: EI - Purged packet from retry buffer. NAK not received as expected.	All
MSGCODE_PCIESVC_DL_RECEIVED_NAK_WRONG_SEQ_NUM ProcessReceivedDLLP: Received NAK for wrong TLP. Packet was not sent with EI.	All
MSGCODE_PCIESVC_DL_ACKNAK_LATENCY_EARLY_TIMEOUT ProcessReceivedDLLP: ACK rcvd early.	All
MSGCODE_PCIESVC_DL_ACKNAK_LATENCY_LATE_TIMEOUT CheckAttachedTimers: ACK/NAK received late.	All
MSGCODE_PCIESVC_DL_UPDATEFC_CREDIT_TIMEOUT TransmitPhy: No VC/FC type Update credits were received within period specified.	All
MSGCODE_PCIESVC_DL_INITFC_CREDIT_TIMEOUT TransmitPhy: All 3 VC InitFC credit types were not received within specified period.	All
MSGCODE_PCIESVC_DL_RECEIVE_TLP_BEFORE_INITFC1 ReceiveTLP: Received TLP on VC before all 3 InitFC1s(P/NP/CPL). FC_INIT is not yet finished.	All
MSGCODE_PCIESVC_DL_RECEIVE_TLP_BEFORE_INITFC2 ReceiveTLP: Received TLP on VC before InitFC2. An InitFC2 is recommended before TLP.	All
MSGCODE_PCIESVC_DL_RECEIVED_DLLP_UNKNOWN_TYPE ProcessReceivedDLLP: Received DLLP with unknown type.	All
MSGCODE_PCIESVC_DL_ACK_NAK_RSVD_FIELD_NON_ZERO CheckDLLPRsvdFields: Received Ack/Nak DLLP with reserved field.	All
MSGCODE_PCIESVC_DL_FC_RSVD_FIELD_NON_ZERO	All

Table B-2 Data Layer protocol compliance checks (Continued)

Protocol Check Name and Description	**Spec Version**
CheckDLLPRsvdFields: Received InitFC with reserved field != 0.	
MSGCODE_PCIESVC_DL_PM_RSVD_FIELD_NON_ZERO CheckDLLPRsvdFields: Received PM DLLP with reserved field !=0.	All
MSGCODE_PCIESVC_DL_RETRY_WRONG_EI_CODE PurgeRetryBuffer: EI - Retry Buffer packet has wrong EI_CODE.	All
MSGCODE_PCIESVC_DL_RETRY_WRONG_SEQ_NUM ProcessReceivedTLP: EI - Expected retry but received retry with wrong seq num.	All
MSGCODE_PCIESVC_DL_IDLE_STARTED_WRONG_LANE ReceivePhy: Idle started on wrong lane.Not implemented.	All
MSGCODE_PCIESVC_DL_IDLE_RECEIVED_IN_MIDDLE_OF_PACKET ReceivePhy: Received IDLE in middle of a packet.	All
MSGCODE_PCIESVC_DL_2_STP_RECEIVED_IN_1_SYMBOL_TIME "ReceivePhy: Received 2 STPs per symbol time, 2nd tlp started wrong lane."""	All
MSGCODE_PCIESVC_DL_STP_RECEIVED_MISSING_END_EDB ReceivePhy: Received STP before previous packet's END/EDB.	All
MSGCODE_PCIESVC_DL_STP_NOT_ON_LANE0_AFTER_IDLE ReceivePhy: Received TLP not aligned to lane 0.	All
MSGCODE_PCIESVC_DL_STP_WRONG_LANE ReceivePhy: Received STP on lane which is not modulo 4.	All
MSGCODE_PCIESVC_DL_2_SDP_RECEIVED_IN_1_SYMBOL_TIME ReceivePhy: Received 2 SDPs per symbol time.	All
MSGCODE_PCIESVC_DL_SDP_RECEIVED_MISSING_END_EDB ReceivePhy: Received SDP before previous packet's END/EDB.	All
MSGCODE_PCIESVC_DL_SDP_NOT_ON_LANE0_AFTER_IDLE ReceivePhy: Received DLLP SDP on lane other than 0. Expected SDP on lane 0.	All
MSGCODE_PCIESVC_DL_SDP_WRONG_LANE ReceivePhy: Received SDP on lane which is not modulo 4.	All
MSGCODE_PCIESVC_DL_PACKET_EXCEEDED_CONTAINER_SIZE ReceivePhy: Packet exceeded SVC container size.	All
MSGCODE_PCIESVC_DL_DATA_RECEIVED_OUTSIDE_PACKET ReceivePhy: Data received outside packet.	All
MSGCODE_PCIESVC_DL_DLLP_SIZE_NOT_8_BYTES	All

Table B-2 Data Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
ReceivePhy: Received DLLP with length which is not 8 bytes.	
MSGCODE_PCIESVC_DL_END_WRONG_LANE ReceivePhy: Received END in wrong lane.	All
MSGCODE_PCIESVC_DL_END_OUTSIDE_PACKET ReceivePhy: Received END outside packet.	All
MSGCODE_PCIESVC_DL_NULLIFIED_DLLP_RECEIVED ReceivePhy: Received nullified DLLP. This is illegal in PCIE.	All
MSGCODE_PCIESVC_DL_EDB_WRONG_LANE ReceivePhy: Received EDB on wrong.	All
MSGCODE_PCIESVC_DL_EDB_OUTSIDE_PACKET ReceivePhy: Received EDB outside packet.	All
MSGCODE_PCIESVC_DL_DLLP_EXCEEDED_8_BYTES ReceivePhy: DLLP exceeded 8 bytes.	All
MSGCODE_PCIESVC_DL_EDB_OUTSIDE_PACKET_8G ReceivePhy: Received errant EDB token on wrong lane or LCRC from previous nullified TLP was bad.	>=3.0
MSGCODE_PCIESVC_DL_EDB_TOKEN_MISSING_8G ReceivePhy: Expected to Receive Phy EDB indication on particular lane.	>=3.0
MSGCODE_PCIESVC_DL_RECEIVED_TLP_FIFO_FULL ProcessReceivedTLP: TLP receive fifo full.	All
MSGCODE_PCIESVC_DL_RECEIVED_DLLP_FIFO_FULL ProcessReceivedDLLP: DLLP received fifo full. Packet will not be queued.	All
MSGCODE_PCIESVC_DL_SENT_DLLP_FIFO_FULL QueueSentDLLP: Sent DLLP fifo full. Packet will not be placed on the sent packet queue.	All
MSGCODE_PCIESVC_DL_CREDIT_SENT_VC_UNINITIALIZED TransmitCredits: Attempting to send credit on uninitialized VC.	All
MSGCODE_PCIESVC_DL_CREDIT_DUPL_INIT TransmitCredits: Init credit already received for VC/fc type.	All
MSGCODE_PCIESVC_DL_RECEIVED_PM_ENTER_L1_DOWNSTREAM "ProcessReceivedDLLP: PM_ENTER_L1 DLLP can only flow upstream, toward root complex."	All
MSGCODE_PCIESVC_DL_RECEIVED_PM_ENTER_L23_DOWNSTREAM "ProcessReceivedDLLP: PM_ENTER_L23 DLLP can only flow upstream, toward root complex."	All
MSGCODE_PCIESVC_DL_RECEIVED_PM_ASPM_REQUEST_L1_DOWNSTREAM	All

Table B-2 Data Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
"ProcessReceivedDLLP: PM_ACTIVE_STATE_REQUEST_L1 can only flow upstream, toward root complex."	
MSGCODE_PCIESVC_DL_RECEIVED_PM_REQUEST_ACK_UPSTREAM "ProcessReceivedDLLP: PM_REQUEST_ACK can only flow downstream, away from root complex."	All
MSGCODE_PCIESVC_DL_RECEIVED_PM_IPG ProcessReceivedDLLP: PM_ENTER_L1 DLLP received with IPG > 4.	All
MSGCODE_PCIESVC_DL_ASPM_L0S_TIMEOUT CheckL0sIdle: ASPM L0s. Phy did not report TX L0s status before timeout occurred.	All
MSGCODE_PCIESVC_DL_ASPM_L1_HANDSHAKE_TIMEOUT ASPMHandshakeSm: ASPM L1 Handshake timeout. SVC Phy not in L1 before timeout occurred.	All
MSGCODE_PCIESVC_DL_ASPM_L1_1_HANDSHAKE_TIMEOUT ASPMHandshakeSm: ASPM L1_1 Handshake timeout. SVC Phy not in L1_1 before timeout occurred.	All
MSGCODE_PCIESVC_DL_ASPM_L1_2_HANDSHAKE_TIMEOUT ASPMHandshakeSm: ASPM L1_2 Handshake timeout. SVC Phy not in L1_2 before timeout occurred.	All
MSGCODE_PCIESVC_DL_PM_L1_HANDSHAKE_TIMEOUT PMHandshakeSm: PM L1 Handshake timeout. SVC Phy not in L1 before timeout occurred.	All
MSGCODE_PCIESVC_DL_PM_L1_1_HANDSHAKE_TIMEOUT PMHandshakeSm: PM L1_1 Handshake timeout. SVC Phy not in L1_1 before timeout occurred.	All
MSGCODE_PCIESVC_DL_PM_L1_2_HANDSHAKE_TIMEOUT PMHandshakeSm: PM L1_2 Handshake timeout. SVC Phy not in L1_2 before timeout occurred.	All
MSGCODE_PCIESVC_DL_PM_L23_HANDSHAKE_TIMEOUT PMHandshakeSm: PM L23 Handshake timeout. SVC Phy not in L23 before timeout occurred.	All

Table B-3 Application Layer protocol compliance checks

Protocol Check Name and Description	Spec Version
MSGCODE_PCIESVC_APPL_TARGET_INVALID_ID ReceivePacket: Bad appl_id received by application.	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_REQ ReceivePacket: Received TLP with unsupported type field.	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_MEM_REQ HandleMemReq: Received unsupported memory request. SVC will return UR.	All

Table B-3 Application Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
MSGCODE_PCIESVC_APPL_TARGET_UNINITIALIZED_MEM_DATA HandleMemReq: MemRd accessed uninitialized memory data.	All
MSGCODE_PCIESVC_APPL_TARGET_HDR_WORD_SIZE_MISMATCH "HandleMemReq: Request has 4 dwords, but addr[63:32] == 0. Request is not a 64-bit memory request."	All
MSGCODE_PCIESVC_APPL_TARGET_DEPRECATED_CFG_REQ HandleCfgReq: Received unsupported (deprecated) Cfg request.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_CFG_TC HandleCfgReq: Received CFG with Illegal Traffic Class - expected 0.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_CFG_DATA_LEN HandleCfgReq: Received CFG with illegal data length - expected 1.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_CFG_AT HandleCfgReq: Received CFG with AT != 2'b00.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_CFG_LAST_BE HandleCfgReq: Received CFG with last DW byte enable != 0.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_IO_TC HandleIOReq: Received IO with illegal Traffic Class - expected 0.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_IO_DATA_LEN HandleIOReq: Received IO with illegal data length - expected 1.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_IO_AT HandleIOReq: Received IO with AT != 2'b00.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_IO_LAST_BE HandleIOReq: Received IO with unexpected last DW byte enable != 0.	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_VENDOR0_MSG "HandleMsgReq: Received unsupported Vendor Defined 0 message, this is a UR."	All
MSGCODE_PCIESVC_APPL_TARGET_UNKNOWN_MSG_CODE HandleMsgReq: Received MSG with unknown message code.	All
MSGCODE_PCIESVC_APPL_TARGET_NONZERO_LAST_BE ByteEnableCheck: Last DW Byte Enable != 0 for length=1.	All
MSGCODE_PCIESVC_APPL_TARGET_ZERO_FIRST_BE ByteEnableCheck: 1st DW Byte Enable == 0 for length > 1.	All
MSGCODE_PCIESVC_APPL_TARGET_ZERO_LAST_BE ByteEnableCheck: Last DW Byte Enable == 0 for length > 1.	All

Table B-3 Application Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
MSGCODE_PCIESVC_APPL_TARGET_BAD_FIRST_BE_NCB_NQA ByteEnableCheck: Bad first DW Byte Enable - non-contiguous bits -- non-QW-aligned.	All
MSGCODE_PCIESVC_APPL_TARGET_BAD_FIRST_BE_NCN_NQA ByteEnableCheck: Bad first DW Byte Enable - non-contiguous with next dword -- non-QW-aligned.	All
MSGCODE_PCIESVC_APPL_TARGET_BAD_LAST_BE_NCB_NQA ByteEnableCheck: Bad last DW Byte Enable - non-contiguous bits -- non-QW-aligned.	All
MSGCODE_PCIESVC_APPL_TARGET_BAD_LAST_BE_NCP_NQA ByteEnableCheck: Bad last DW Byte Enable - non-contiguous with prev dword -- non-QW-aligned.	All
MSGCODE_PCIESVC_APPL_TARGET_BAD_FIRST_BE_NCB ByteEnableCheck: Bad first DW Byte Enable - non-contiguous bits.	All
MSGCODE_PCIESVC_APPL_TARGET_BAD_LAST_BE_NCB ByteEnableCheck: Bad last DW Byte Enable - non-contiguous bits.	All
MSGCODE_PCIESVC_APPL_TARGET_POISON_CFG_REQ HandleCfgReq: Received CFG with Poison bit set. SVC will return UR.	All
MSGCODE_PCIESVC_APPL_TARGET_INVALID_64_BIT_ADDR HandleMemReq: Received Unsupported 64-bit read request. SVC will return Completer Abort status	All
MSGCODE_PCIESVC_APPL_TARGET_POISON_IO_REQ HandleCfgReq: Received IOWr with Poison bit set. SVC will return UR.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_MSG_DATA_LEN HandleMsgReq: Received MSG with reserved data length field !=0.	All
MSGCODE_PCIESVC_APPL_TARGET_MSG_INTX_ON_DOWNSTREAM_PORT HandleMsgReq: Received INTx on downstream port.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_MSG_AT HandleMsgReq: Recevied MSG TLP with AT != 2'b00.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_MSG_RSVD_HDR "HandleMsgReq: Reserved vendor defined msg has routing by addr (illegal) [2.2.8.6], hdr[2] == 0, or hdr[2] == 0."	All
MSGCODE_PCIESVC_APPL_TARGET_UNEXPECTED_TYPE_1_CFG HandleCfgReq: Received unexpected Type 1 Cfg request. SVC will return UR.	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_ATOMIC_OP_REQ HandleMemReq: Received Illegal atomic-op request. Atomic Ops not supported in PCIE version < 2.1. Svc will return UR.	<=2.0
MSGCODE_PCIESVC_APPL_TARGET_CFG_INVALID_FUNC_NUM HandleCfgReq: Received CfgRd with Invalid Function Number. SVC will return UR.	All

Table B-3 Application Layer protocol compliance checks (Continued)

Protocol Check Name and Description	Spec Version
MSGCODE_PCIESVC_APPL_TARGET_CFG_INTERNAL_PROBLEM HandleCfgReq: Received CfgRd with invalid function. Was SetCompleterID(0x%0x) set for this target? (returning CA)	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_VENDOR1_MSG HandleMsgReq: Received unsupported Vendor Defined 1 message.	All
MSGCODE_PCIESVC_APPL_TARGET_POISON_MEM_WR HandleMemReq: Poison bit set on received MemWr. SVC will return UR.	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_PREFIX Unsupported TLP prefix type received. Discarding malformed TLP.	>=2.1
MSGCODE_PCIESVC_APPL_TARGET_MAX_SUPPORTED_PREFIX TLP contains too many end-end TLP prefixes.	>=2.1
MSGCODE_PCIESVC_APPL_DRIVER_COMMAND_TIMEOUT CheckForCommandTimeout: Request with a particular tag has timed out.	All
MSGCODE_PCIESVC_APPL_DRIVER_INVALID_MEM_DATA CheckReceivedData: Data uninitialized at shadow addr.	All
MSGCODE_PCIESVC_APPL_DRIVER_DATA_MISMATCH CheckReceivedData: Data mismatch at shadow addr.	All
MSGCODE_PCIESVC_APPL_DRIVER_LOW_BYTE_COUNT Byte count field is less than expected value. Discarding completion.	All
MSGCODE_PCIESVC_APPL_DRIVER_HIGH_BYTE_COUNT Byte count field is greater than expected value. Discarding completion.	All
MSGCODE_PCIESVC_APPL_DRIVER_MISSING_GOOD_STATUS ProcessReceivedPacket: Received completion with unexpected status.	All
MSGCODE_PCIESVC_APPL_DRIVER_UNEXPECTED_GOOD_STATUS "ProcessReceivedPacket: Received completion status of GOOD, but expected status other than GOOD."	All
MSGCODE_PCIESVC_APPL_DRIVER_BAD_RECD_LEN CheckReceivedData: TLP received length != expected length in bytes	All
MSGCODE_PCIESVC_APPL_DRIVER_SPURIOUS_CPL Spurious completion received.	All
MSGCODE_PCIESVC_APPL_DRIVER_MISMATCHED_CPL_ATTR Completion attr does not match request attr. Discarding completion	All
MSGCODE_PCIESVC_APPL_DRIVER_MISMATCHED_CPL_TC Completion TC does not match request TC. Discarding completion	All

Table B-3 Application Layer protocol compliance checks (Continued)

Protocol Check Name and Description	**Spec Version**
MSGCODE_PCIESVC_APPL_REQUESTER_UNINITIALIZED_MEM_DATA CheckReceivedData: Data uninitialized at shadow addr.	All
MSGCODE_PCIESVC_APPL_REQUESTER_DATA_MISMATCH CheckReceivedData: Data mismatch at shadow addr.	All
MSGCODE_PCIESVC_APPL_REQUESTER_BAD_RECD_LEN CheckReceivedData: Bad received length != expected length in bytes.	All
MSGCODE_PCIESVC_APPL_REQUESTER_BAD_RECD_BYTE_CNT HandleMemCpl: Received byte_cnt != expected byte_cnt.	All
MSGCODE_PCIESVC_APPL_REQUESTER_COMMAND_TIMEOUT ReceivePacket: Request has timed out waiting for Cpl.	All
MSGCODE_PCIESVC_APPL_REQUESTER_SPURIUS_CPL HandleMemCpl: Spurious Cpl packet received. Potentially late Cpl for timed-out request.	All
MSGCODE_PCIESVC_APPL_REQUESTER_TIMEOUT_BEYOND_RETRY_COUNT "ReceivePacket: Timed out request, retry_count is beyond max retry count - failing request."	All
MSGCODE_PCIESVC_APPL_REQUESTER_TIMEOUT_MISSING_CPL MarkTimedOutRequests: Timeouts caused some missing Cpl.	All

C
PCIe UVM Interface

The PCIe VIP is implemented as a set of Verilog and SystemVerilog modules that provide control variables and tasks to interface with them and control their behavior. UVM verification environments, on the other hand, are comprised of class-based components connected together with TLM ports. The UVM Interface is a bridge between the PCIe VIP and the user's test bench.

The UVM interface is not a new implementation of the VIP. It is an adapter API that works together with the existing models. Users instantiate the module for the VIP instantiation model, then instantiate the UVM component to connect with the module. To enable the UVM interface, define the macros PCIESVC_INCLUDE_UVM_API and UVM_PACKER_MAX_BYTES=8192, and compile the SVC modules.

The UVM interface uses the SVCVerilog API to communicate with the VIP. Only one of the Verilog API, SystemVerilog API, and UVM API can be used at a time. Using more than one API simultaneously results in unpredictable behavior.

C.1 Compiling the UVM Interface

The UVM is an open source code library written in SystemVerilog by Accellera Systems Initiative. Synopsys does not provide a copy of the UVM. It can be obtained from http://uvmworld.org. The Synopsys PCIe UVM Interface requires UVM version 1.0p1 or greater.

To enable the UVM interface, define the macro `PCIESVC_INCLUDE_UVM_API` and compile the SVC modules. Enabling the UVM interface introduces SystemVerilog code into the SVC module files. These files have .v file name extensions. If the simulator uses filename extensions to determine whether or not to enable SystemVerilog features it may be necessary to explicitly tell the simulator to compile these files with SystemVerilog features turned on. Simulators provide a command line option for this purpose.

The UVM interface is contained inside of a SystemVerilog package named `pciesvc_uvm_pkg`. This package is located in the file Verilog/Include/pciesvc_uvm_pkg.sv in the source tree. The package `` `includes `` files from the other model directories so these directories must be added to the include path (`+incdir+`*`dir`*).

The UVM API for the instantiation models is not part of the package. The model UVM components are located at Verilog/InstantiationModels/. Two files, pciesvc_device_uvm_component.sv and pciesvc_mac_uvm_component.sv are required.

C.2 UVM Component Construction and Instantiation

The UVM interface is constructed from `uvm_components`. The component hierarchy exactly parallels the module hierarchy of the instantiation model. For each model in the Verilog/InstantiationModels/ directory there is a matching `uvm_component` which instantiates all the child components for each of the individual SVCs. The SVC-level components instantiate their own children, `uvm_agents`, which are the interface to the SVC module. The `uvm_components` that you instantiate are base classes whose purpose is to provide a class handle in the environment. The base classes are inherited by classes located inside the SVC modules. These derived classes implement the behavior of the UVM API. An instance override is registered with the UVM factory so that when the top-level `uvm_component` is instantiated, the implementation classes are created. The SVC takes care of applying the instance override so you do need to worry about creating the correct class types. You only need to to tell the UVM API component the module instance scope of the corresponding SVC instantiation model. The component is told this information by setting a variable named `model_instance_scope` in the `uvm_config_db #(int)`. For example, if the instantiation model module is instantiated like this:

```
module top;
  ...
  pciesvc_device_serdes_x4_model  root0( … );
```

then the model instance scope is top.root0. In the build phase, set the model_instance_scope by calling `set_config_int` like this:

```
pciesvc_device_ uvm_component root0;

virtual function void build_phase( uvm_phase phase );
  super.build_phase( phase );

  set_config_string( "root0", "model_instance_scope", "top.root0" );
  root0 = new( "root0", this );
```

C.2.1 UVM Component Organization

Each SVC module has a corresponding UVM component whose class name follows the naming convention pciesvc_*layer name*_component. The SVC component instantiates multiple other UVM components. These are generally one of two types – `uvm_agent` or standalone `uvm_monitor`.

Each `uvm_agent` contains a `uvm_driver` and a `uvm_sequencer`. The agents typically do not contain a `uvm_monitor` because they are connected to SVC modules, not the pins of a DUT, so they do not capture low level signals that must be encoded into transactions.

The SVC does, however, generate spontaneous output information. A standalone `uvm_monitor` is instantiated in the SVC-level component to communicate this kind information to the UVM environment. In general, if a SVC contains a Verilog event that triggers on some condition, the UVM API will contain a monitor to wait for that event and generate a transaction for it.

Components in the SVC modules have predictable names based on naming conventions. The names of component classes are based on the names of the sequence items the component uses. The class names of agents are usually pciesvc_*layer name_transaction name*_agent. The class names of monitors are usually pciesvc_*layer name_transaction name*_monitor. For example, a sequence item named “pciesvc_dl_control_item” will execute on an agent named “pciesvc_dl_control_agent”. The instantiation name for components is the class name with the “pciesvc_*layer name_”* prefix removed. For example, the component `pciesvc_dl_control_agent` will be instantiated with the name `control_agent` inside the class `pciesvc_dl_component`.

Within a SVC agent, the `uvm_driver`, `uvm_sequencer`, and `uvm_monitor` are always instantiated with the names driver, sequencer, and monitor, respectively. Within a standalone monitor, the analysis port is always named `monitor_port`. For convenience, all analysis ports are exported up to the containing SVC-level component. The instance name of the analysis port at that level follows the convention *monitor name*_port. So if the standalone monitor's instance name is link_state_monitor then the analysis port in the SVC component is named link_state_port.

The description for each transaction in "Accessing Constraints and Statistics" will indicate the name of the SVC component that uses the transaction and any exceptions to the naming rules will be described there. The UVM can show all the class names and instantiation names for the entire API by calling the `print` function of the instantiation model component.

C.2.2 Phasing

The UVM interface uses the run phase, not the new UVM phasing schedule. Users are responsible for raising an objection during the run phase and clearing the objection when the test has finished applying stimulus.

The PCIe VIP raises objections during the run phase when it is busy. It will drop the objection when it becomes idle again. The SVC whose UVM component raises the objection has a task call in its Verilog API for waiting until the SVC is idle. These SVCs are the Link Layer, Transaction Layer, Requester Application, and Driver Application. UVM tests can check for the raised objection to find out if that SVC is idle or not. One way to accomplish this is as follows:

```
virtual task run_phase( uvm_phase phase );
  uvm_object objectors[$];
  ...

  phase.get_objectors( objectors );
  foreach ( objectors[ i ] ) begin
    if ( objectors[ i ].get_name == "dl0" ) begin
      // dl0 is not idle.
    end
  end
```

C.2.3 Logging

These are the items that the UVM API will print to the log file at each verbosity level:

UVM_NONE	Only error messages will be printed.
UVM_LOW	Important messages which are not error messages.
UVM_MEDIUM	Drivers and monitors print transactions they transmit and receive.
UVM_HIGH	More detailed messages about component operation.
UVM_FULL	Internal debug messages, only useful for Synopsys support.

The UVM hierarchical reporting interface only controls the logging of the PCIe UVM API. The base PCIe VIP uses the Message Log facility and is not affected by the UVM controls. Refer to "Message Logging Facility" on page 239 file for more information about Message Log.

Depending on the verbosity levels, information can be duplicated by the UVM API and the Message Log. To avoid this duplication, set the verbosity of the UVM API to UVM_LOW. Do not turn off Message Log logging and rely only on the messaging from the UVM API. The UVM API does not print enough information at any verbosity level to be a reliable debugging tool.

C.3 Accessing Constraints and Statistics

Constraint VARs and statistics are accessed using the register abstraction layer. A `uvm_reg_block` is created for the `pciesvc_device_uvm_component` and stored in the configuration resource database. The reg block class is named `pciesvc_device_vars`. It is defined in the same file as the model component, pciesvc_device_uvm_component.sv.

The reg block contains more reg blocks inside it, one for each protocol layer. The hierarchy of reg blocks matches the UVM model component and the instantiation model. The same names match as well.

The name of the variable in the configuration database is `sassvc_vars`. The following code section shows how the reg block can be retrieved from the config database. Change the arguments to `get()` function as appropriate for your environment.

```
uvm_config_object config_db_item;
pciesvc_device_vars root0_vars;

status = uvm_config_db #(uvm_object)::get(
  this,
  "root0",
  "pciesvc_vars",
  config_db_item
);
if ( status ) begin
  if ( ! $cast( root0_vars, config_db_item ) )  begin
    ERROR.
  end
end
else begin
  ERROR.
end
```

The reg block for each protocol layer SVC is in the *_uvm_api.sv file. Its class name is of the form pciesvc_*SVC name*_vars. This class contains a `uvm_reg` instance for each VAR constraint in the SVC. The name of the `uvm_reg` instance is the same as the VAR name. If the SVC contains other SVCs as subcomponents, the reg block also has reg block members for the contained SVCs.

C.3.1 Setting Constraints and Reading Statistics

Setting constraints is done by calling the `write` method of the `uvm_reg` instance for a particular VAR. Only the `status` and `value` arguments should be specified. All other arguments should be left at their default values.

```
uvm_status_e status;
root0_vars.driver[0].min_time_to_next_transaction_var.write(
  status,50
);
```

That status can be checked if desired. Registers are accessed using the backdoor access method and accesses consume no simulation time so they should always complete with a status of UVM_IS_OK.

Statistics are accessed similarly with the `read` method. Statistics are read only. Calling the `write` method for a statistics register generates an error message.

```
int num_tlp_sent;
root0_vars.port[0].dl[0].stat_num_tlp_sent.read(status,num_tlp_sent);
```

C.3.2 Important Notes for the Register Abstraction Layer

Even through the Register Layer is used, Synopsys VARs are not registers like those that would normally be found in an RTL DUT. There is no register bus. All access happens in zero time using the backdoor access method. Consequently, only a subset of RAL features are relevant to the Synopsys UVM API. The following rules should be observed:

- Only the `read` and `write` methods should be used to access registers. Other methods, including but not limited to, `get`, `set`, `peek`, and `poke` are not implemented and should not be used.
- VARs are not memory mapped. The `uvm_reg_maps` are not visible to users. Register map operations are not implemented and may lead to undefined behavior.

C.4 UVM Sequence Items

You manipulate the SVC by exchanging `uvm_sequence_items` with the various `uvm_agents` that make up the UVM interface. In general, the sequence items encapsulate the arguments for one or more of the SVC tasks. The agents will accept sequence items and call the corresponding task. Some sequence items are more complex and encapsulate more abstract SVC operations. Multiple task calls will be executed for such sequence items.

These sequence items described in this section are the low level interface for communicating with the SVC. Creating more interesting sequences out of the sequence items is an application-specific problem that falls outside the scope of this document. Some commonly used sequences might be provided and are described in another section.

The transactions for each SVC model are located in the directory for that model in a file that ends with *_uvm_transactions.sv. Refer to these files for the transaction item definitions. Some SVCs do not have any sequence item so the uvm_transactions file will not exist or will be empty.

When some sequence items are sent to the `uvm_driver`, a response is returned back to the sequencer on the response (RSP) port. If the response type is the same as the request type the response is a modified copy of the request. The original request is never modified by the agent.

All sequence items implement the basic `uvm_object` methods `print`, `copy`, `clone`, `compare`, and so on. The `pack` and `unpack` methods are implemented, but since sequence items are not converted to a bus interface these tasks are not useful.

C.4.1 Control Agent Sequence Items

The PCIe VIP has a control agent for controlling its behavior. There are multiple control agent sequences, all inherited from a common base class. The base class has a name like pciesvc_*SVC name*_control_item. There is a derivative class for each control task which the agent handles. The name of the per-task class is derived from the task name using the following naming convention: Convert CamelCaseTaskNames to underscore_separated_names, then prefix “pciesvc_*SVC name*_” and postfix “_item”. So task DisplayStats becomes pciesvc_dl_display_stats_item, for example.

Each control item contains data members for each of the arguments that the task requires. When a task takes no arguments the class contains no public data members. When a task has output arguments, those values are returned from the driver by copying the sequence item, filling in the output values, and returning the copy on the sequencer's response (RSP) port.

When the control agent's driver receives a control item it executes the corresponding task call. Tasks that consume simulation time will block the driver from consuming more items (and executing more tasks) until the task completes.

C.4.2 Driver Application

C.4.2.1 pciesvc_driver_transaction_item

This sequence item is used to send transactions to the Driver Application. It covers the functionality of the `Queue*()` tasks. Set the `transaction_type` field to configure the type of transaction to queue. Valid values for this field are defined in the `pciesvc_tl_type_enum` in the Verilog/Include/pciesvc_uvm_parms.svh file. There are fields in the sequence item that have the same name as the arguments to the task calls. Fill in the necessary fields for the transaction type procedurally or with the constraint solver. For performance reasons, the dynamic array `payload_data` is not a random variable. The `post_randomize` method of the `pciesvc_driver_transaction_item` is implemented to allocate `payload_data` with a size to match the `length` field and fill in each dword with an incrementing pattern. If you do not need automatic payload generation because you have an application specific payload to put in `payload_data` after randomization, you can gain a further performance improvement by setting the `randomize_payload` field to 0. This prevents `post_randomize` from generating `payload_data` that will be clobbered by user data. Execute the sequence item on the `pciesvc_driver_queuing_agent` in the `pciesvc_driver_component`.

The `uvm_driver` for the queuing agent will return a response to the sequencer. The `block` field controls whether the `uvm_driver` will return immediately when the transaction is queued or block waiting for a completion before completing the item. If `block == 1` then the completion, if any, will be returned in the `completion` field. The `completion` field is a class handle of type `pciesvc_driver_completion_item`. Completion items contain the completion status and any data associated with the completion. If `block == 0` then the completion field is unmodified and the `command_num` field contains a valid command number. (The command number is not the same as a TLP sequence number.)

A `pciesvc_driver_transaction_item` with a valid `command_num` can be used to pick up completions by executing it on the `pciesvc_completion_agent`. The completion agent will create the response and fill in the `completion` field. The `uvm_driver` returns the response to the sequencer. If `transaction_item` is sent to the completion agent with an invalid `command_num` or if the completion has already been picked up, the agent will issue a warning message.

C.4.3 Completion Target Application

C.4.3.1 pciesvc_target_appl_mem_write_item

This sequence item is returned by the `pciesvc_target_appl_mem_write_notification_monitor` when a memory write notification event occurs. This item is a monitor output only – there is no sequencer or driver on which to execute this item. This item covers the functionality of the `PopMemWriteNotification` task and the `event_mem_write` event.

The `PopMemWriteNotification` task has an output argument named `valid` which the task sets to 1 when the other output arguments contain valid data. There is no corresponding `valid` field in the sequence item. The monitor takes care of checking the valid bit and does not generate an item if valid equals 0. The monitor reads the data out of memory and returns it in the data array member.

The `MEM_WRITE_NOTIFICATION_MODE_VAR` variable must be set to a nonzero value before the monitor will generate any items.

C.4.3.2 request_tlp_port

The `pciesvc_target_appl_component` has an analysis port, `request_tlp_port`. All TLPs that reach the target application are broadcast out this analysis port as `pciesvc_tlp_item` sequence items (see "pciesvc_tlp_item"). All TLP fields of inbound requests are accessible via this interface. It is recommended to use the `pciesvc_target_appl_mem_write_notification_monitor` instead unless the additional TLP information is required.

C.4.4 Memory Target Application

C.4.4.1 pciesvc_mem_target_write_item

This item is for writing the memory used by the memory target application. It can be used to pre-populate the memory with data.

The driver will call `PreWriteHint` to pre-allocate a memory range in the target memory. Then it will copy the `data[]` array into the allocated memory space. Users do not usually need to perform a `PRE_WRITE_HINT` (with the `pciesvc_mem_target_pre_write_hint_item`) operation before doing a write. The API does this internally. The length of the `data` dynamic array is the number of dwords that will be written.

C.4.4.2 pciesvc_mem_target_read_item

For read operations, set the `read_data_length` field to the number of dwords to read. The read response is returned on the sequencer's response (RSP) port. The response is a copy of the request with the `data[]` array allocated to match `read_data_length` and filled in read data.

C.4.5 I/O Target Application

C.4.5.1 pciesvc_io_target_write_item

Analogous to the above Mem Target, this item is for writing the memory space used by the I/O target application. It can be used to pre-populate that space with data.

The I/O Target is written one dword at a time to the I/O address specified.

C.4.5.2 pciesvc_io_target_read_item

For I/O read operations, the read response is returned on the sequencer's response (RSP) port. The response is a dword filled in read data.

C.4.6 Transaction Layer

C.4.6.1 pciesvc_tl_vc_config_item

This sequence item is for setting up, enabling, and disabling VCs in the transaction layer. It covers the functionality of the `SetTrafficClassMap`, `SetInitTxCredits`, `SetVcEnable`, and `IsVcInitFinished` tasks. These sequence item is executed on the `pciesvc_tl_vc_config_agent` in the `pciesvc_tl_component`.

The sequence item has fields that map to the arguments of the various tasks. When the driver gets the sequence item from the sequencer it will read the fields and call `SetInitTxCredits`, `SetTrafficClassMap`, then `SetVcEnable`. To configure a VC, set the enable field to 1, fill in values for the other fields and execute the item. The field `tcs_to_map_to_vc` is a bit field which specifies which TCs will be mapped to the VC.

Only bits 0 through 7 are used. If the field `block_until_vc_init_complete` field equals 1 the driver will not complete the item and return control to the sequencer until the VC flow control initialization process has completed for the VC.

It is possible to change the TC to VC mapping after a VC has been initialized. In the `pciesvc_tl_vc_config_item`, set the `enable` field to 1 and set `tcs_to_map_to_vc` to the desired mapping. The driver will not set initial Tx credit values or re-enable the VC after it is already initialized. It is not recommended to change the traffic class mapping when there are unacknowledged TLPs outstanding or outstanding requests whose completions have not arrived.

C.4.6.2 pciesvc_tlp_item

This sequence item models a complete TLP. It can contain any kind of TLP, either a request or a completion. It is fully randomizable and may be overridden in the factory to add user-defined constraints. All the sequence item convenience functions are implemented (`print`, `copy`, `pack`, `unpack`, `compare`, and so on). `pciesvc_tlp_item` is in `pciesvc_uvm_pkg`. It is used by the Transaction Layer and multiple applications.

The data members in `pciesvc_tlp_item` match the field names in the PCIe specification. The data members that are relevant for any kind of TLP are the ones listed in the specification for that TLP type.

C.4.6.3 TL Events

There are four events in the Transaction Layer SVC that are associated with sending and receiving TLPs and credits; `event_sent_credit`, `event_received_credit`, `event_sent_tlp`, and `event_received_tlp`. There is no data associated with these events and the TLPs and credits cannot be retrieved from the TL. (See the Link Layer API for ways to retrieve sent and received TLPs. See the Transaction Layer `Query*` tasks for retrieving credit values.) There is a standalone monitor in the transaction layer component for each of these events. Their names are `event_sent_credit_monitor`, `event_received_credit_monitor`, `event_sent_tlp_monitor`, and `event_received_tlp_monitor`. The data type for the monitor port is `uvm_event`.

C.4.7 Data Link Layer

C.4.7.1 pciesvc_dl_tlp_monitor_item

This sequence item is returned by the `pciesvc_dl_sent_tlp_monitor` when a sent TLP event occurs and it is returned by the `pciesvc_dl_received_tlp_monitor` when a received TLP event occurs. This item is a monitor output only; there is no sequencer or driver to execute this item on. This item covers the functionality of the `SentTLP` and `ReceivedTLP` tasks and the events `event_sent_tlp` and `event_received_tlp`.

When a monitor generates one of these sequence items, the contents of the TLP is stored in the dword `tlp` array. The length of the array (`tlp.size()`) is the number of dwords.

The `SentTLP` and `ReceivedTLP` tasks have an output argument named `valid` which the tasks set to 1 when the other output arguments contain valid data. There is no corresponding `valid` field in the sequence item. The monitor takes care of checking the `valid` bit and does not generate an item if `valid` equals 0.

The `SENT_TLP_INTERFACE_MODE_VAR` and `RECEIVED_TLP_INTERFACE_MODE_VAR` variables must be set to a nonzero value before the monitors will generate any items.

C.4.7.2 pciesvc_dl_dllp_monitor_item

This sequence item is returned by the `pciesvc_dl_sent_dllp_monitor` when a sent DLLP event occurs and it is returned by the `pciesvc_dl_received_dllp_monitor` when a received DLLP event occurs. This item is a monitor output only; there is no sequencer or driver on which to execute this item. This item covers the functionality of the `SentDLLP` and `ReceivedDLLP` tasks and the `event_sent_dllp` and `event_received_dllp` events.

When a monitor generates one of these sequence items, the contents of the DLLP is stored in the dword array named `dllp`. The length of the array (`dllp.size()`) is the number of dwords.

The `SentDLLP` and `ReceivedDLLP` tasks have an output argument named `valid` which the tasks set to 1 when the other output arguments contain valid data. There is no corresponding `valid` field in the sequence item. The monitor takes care of checking the `valid` bit and does not generate an item if `valid` equals 0.

The `SENT_DLLP_INTERFACE_MODE_VAR` and `RECEIVED_DLLP_INTERFACE_MODE_VAR` variables must be set to a nonzero value before the monitors will generate any items.

C.4.7.3 Link State Transaction

The link state indication is a Boolean value which reflects whether the DL is indicating “link up” or “link down” to the transaction layer. There is no sequence item to represent the link state. The transaction type is the native type “bit.” The standalone monitor `pciesvc_dl_link_state_monitor` generates link state transactions when the link state changes and the value of the bit indicates the new link state.

C.4.8 Physical Layer and LTSSM

C.4.8.1 LTSSM State Transaction

The current LTSSM state is reported to the UVM environment by the `pciesvc_phy_ltssm_state_monitor`. There is no sequence item to represent the LTSSM state. The transaction type is the native type “int.” The monitor generates transactions when the LTSSM state changes and the value of the int indicates the new link state. The set of possible values is defined by the `STATE_LTSSM_*` parameters in pciesvc_phylayer_parms.v

C.4.8.2 Link State Transaction

The link state indication is a Boolean value which reflects whether the PL is indicating “link up” or “link down” to the data link layer. There is no sequence item to represent the link state. The transaction type is the native type “bit.” The standalone monitor `pciesvc_phy_link_state_monitor` generates link state transactions when the link state changes and the value of the bit indicates the new link state.

C.5 Global Shadow

The `pciesvc_global_shadow` is not contained in the instantiation model. It gets instantiated separately, one in each PCIE root hierarchy. Likewise, the `pciesvc_global_shadow_component` is not contained in the instantiation model component and must be instantiated separately in the UVM environment.

The method for creating the global shadow component is similar to the way the model component is created. There is a string configuration database variable to set which connects the UVM component to the corresponding SVC module. The name of this variable is `global_shadow_instance_scope`.

The global shadow component must be created with the factory. The `context` argument of the create method must be the full hierarchical scope of the module where the `pciesvc_global_shadow` SVC in instantiated. For example, if the global shadow SVC is instantiated in module “top” like this:

```
module top;
  ...
  pciesvc_global_shadow  global_shadow0();
```

then the global shadow UVM component should be created like this:

```
pciesvc_global_shadow_component   global_shadow0;
  virtual function void build_phase( uvm_phase phase );
  super.build_phase( phase );

  set_config_string(
    "global_shadow0",
  "global_shadow_instance_scope",
  "top.global_shadow0"
  );
  global_shadow0 =
  pciesvc_global_shadow_component::type_id::create(
    "global_shadow0",
    this,
    "top"
  );
```

The global shadow component has its own register block for its VARs. The reg block is stored in the config database under the variable name `global_shadow_vars`.

In addition, the global shadow component also contains subcomponent instantiations of a `pciesvc_mem_target_component` (`shadow_mem[ ]`) and a `pciesvc_cfg_database_component` (`shadow_cfg[ ]`) as well as the usual per-component control agent (`control_agent`).

C.6 TL Internal Interface to Application Layer

The Transaction Layer SVC has a public interface to the application layer. This allows users to write their own application and configure the TL to route TLPs to that application. Refer to the `SendPacket` and `ReceivePacket` tasks in "Transaction Layer Internal Interface Tasks". The UVM API recasts this interface as a pair of TLM ports and a UVM agent named `pciesvc_user_appl_agent`.

As background information, all applications, whether user-defined or shipped by Synopsys, have an application ID. The TL SVC maintains mapping tables so that it can route TLPs to an application-based the ID. The driver, requester, and target applications all have default application IDs assigned. The default mappings in the TL are set up automatically so the applications work as expected. IDs must be unique among all applications in a PCIe device. Part of writing a user application is choosing an application ID and configuring the TL to route requests to the application. Refer to the `Add*ApplIdMapEntry` tasks in "Transaction Layer User Tasks" for more information about request TLP routing.

C.6.1 TL TLM Ports

The `pciesvc_tl_component` has a pair of TLM ports for exchanging TLPs with an application. The `receive_packet_port` is a is a nonblocking put port that passes received TLPs to an application. The `send_packet_port` is a nonblocking get port that retrieves outbound TLPs from an application for queuing in the TL. Both, `receive_packet_port` and `send_packet_port` are associative arrays, indexed by application ID.

C.6.2 pciesvc_user_appl_agent

This class is a UVM agent with a sequencer, driver and monitor. It is the base for user applications; the application is created by executing sequences on the sequencer. The agent classes are defined in Include/ pciesvc_uvm_tasks.sv

The `pciesvc_user_appl_agent` has a pair of TLM export, `receive_packet_export` and `send_packet_export`. These ports are connected to the TLM ports in the TL by the device model. The monitor is in the receive path, receiving inbound packets, which are instances of `pciesvc_tlp_item`. Its analysis port is connected to the `receive_packet_analysis_port` in the agent for other testbench components to connect with and it is connected back to the sequencer's `receive_packet_imp` port so that application sequences have access to received packets. The `receive_packet_imp` is implemented to put received packets in the `received_packet_mbox` , which is an unbounded mailbox. Sequences should query the mailbox to pick up received packets. The sequencer passes `pciesvc_tlp_items` to the driver, which feeds them out the `send_packet_export`.

The `pciesvc_user_appl_agents` are instantiated in `pciesvc_device_uvm_component`. The `user_appl_agent` variableis an associative array, indexed by application ID of `pciesvc_user_appl_agents`. By default, the array is empty. You must add applications to it as described in "Configuring the UVM API for User Defined Applications".

As a rule, you should not override Synopsys UVM components using the factory. The `pciesvc_user_appl_agent` is an exception, however. The agent and its monitor, driver, and sequencer subcomponents are registered with the factory. Factory overrides can be provided if it is desirable to augment or replace these components.

C.6.3 Configuring the UVM API for User Defined Applications

The first step in defining a UVM-based application is selecting an application ID. Application IDs must be unique for all application instances in a device. To see which IDs are used by Synopsys applications look in Verilog/Application_PCIE/pciesvc_application_parms.v. The parameters that start with APPL_ID_ are preconfigured for use with existing applications. It is possible to change these default configurations but it is easier to pick a non-colliding value. The minimum value is 0. the maximum value is one less than the `NUM_APPL_IDS` parameter in Verilog/TransactionLayer/pciesvc_transaction.sv. `NUM_APPL_IDS` is 8 so the largest application ID that could be used, by default, is 7.

The device UVM component needs to be told what the application IDs of the user defined applications are. This is done with the `uvm_config_db` and the `pciesvc_user_appl_config_object`, as shown in the code example below. For each application, add its application number to the `user_appl_ids` queue. Synopsys applications are already registered and should not be added to the queue. Set the `config` object in the configuration database. Do this in the build_phase. The UVM component path should be the path to the `pciesvc_device_uvm_component` instance, env.endpoint0 in this example. The configuration variable is `user_appl_config`.

```
pciesvc_user_appl_config_obj  appl_cfg_obj;

appl_cfg_obj = new( "appl_cfg_obj" );
appl_cfg_obj.user_appl_ids[0] = 3;
set_config_object(
  "env.endpoint0", "user_appl_config", appl_cfg_obj
);
```

This causes an instance to be added to the `user_appl_agent` array and its TLM ports connected to the transaction layer for each registered application ID.

Setting up the mapping to route requests transactions to the new application is done in the TL by executing control items. If the application will never act as a completer, which handles inbound requests, then it is not necessary to set up any mappings. Inbound completions are always routed back to the application based that originated the request. This is done based on the Requester ID. Therefore, the requester ID should not be the same as another application.

There are several control items for mapping requests by ID, memory address, message code, or other TLP properties to an application ID. The following is a code example that maps memory requests in a certain memory range to the application defined above. Do this during the run phase.

```
pciesvc_tl_add_mem_addr_appl_id_map_entry_item  addr_appl_id_map;

addr_appl_id_map = new( "addr_appl_id_map" );
addr_appl_id_map.memory_addr    = my_mem_range_min_addr;
addr_appl_id_map.memory_window  = my_mem_range_max_addr –
                                  my_mem_range_min_addr + 1;
addr_appl_id_map.appl_id        = 3;
env.endpoint0.port[0].tl[0].addr_appl_id_map_agent.sequencer.
                                      execute_item( addr_appl_id_map );
```

At this point the new application is available to handle requests. A sequence should be developed for the application agent's sequencer to implement the application's behavior.

D
PCIe PIE-8 Interface

The PIE-8 specification is the *"PHY Interface Extensions Supporting 8GT/s PCIe*" (Revision 2.02 dated October 1, 2014). It is an extension of the PIPE 2.0 specification that supports 8.0GT/s and 16.0GT/s data rates and equalization at those rates.

Attention Synopsys supports only that portion of this PIE-8 specification as it relates to passing information to and from the PHY for equalization control and response.

D.1 Supported Interface Signals

Table D-1 lists the PIE-8 signals implemented to support equalization functionality.

Table D-1 **PIE-8 Control/Response Signals**

Name	Direction	Active Level	Description
MacDataEn	Input	High	Used for 8.0 GT/s and 16.0 GT/s equalization and setting Lane numbers. When '1', a transfer to the PHY is in-progress. One signal per Lane.
MacData[5;0]	Input	N/A	Used with MACDataEn for 8.0 GT/s and 16.0 GT/s equalization and setting Lane numbers. Control and data for equalization control and setting Lane numbers. One signal per Lane.
PhyDataEn	Ouput	High	Used for 8.0 GT/s and 16.0 GT/s equalization. When '1', a transfer to the MAC is in-progress. One signal per Lane.
PhyData[5:0]	Ouput	N/A	Used with PHYDataEn for 8.0 GT/s and 16.0 GT/s equalization. Control and data for equalization control. One signal per Lane.

All other signals are the same as the PIPE 4.3 specification (PCLK as PHY output) with the exception of those signals specified in the following table.

Table D-2 PIPE 4.3 Equalization Signals Not Used

Name	Spipe Dir.	Lane 0 Mpipe / Spipe Name	Mpipe / Spipe Task
GetLocalPresetCoefficients			
	Input	get_local_preset_coefficients_0 / attached_get_local_preset_coefficients_0	PipeSerdesEqControl / PipeSlaveEqControl
LocalTxCoefficientsValid			
	Output	local_tx_coefficients_valid_0 / attached_local_tx_coefficients_valid_0	PipeSerdesEqControl / PipeSlaveEqControl
LocalPresetIndex[3:0]			
	Input	local_preset_index_0 / attached_local_preset_index_0	PipeSerdesEqControl / PipeSlaveEqControl
LocalTxPresetCoefficients			
[17:0]	Output	local_tx_preset_coefficients_0 / attached_local_tx_preset_coefficients_0	PipeSerdesEqControl / PipeSlaveEqControl
LocalFS[5:0]			
	Output	local_fs_0 / attached_local_fs_0	PipeSerdesEqControl / PipeSlaveEqControl
LocalLF[5:0]			
	Output	local_lf_0 / attached_local_lf_0	PipeSerdesEqControl / PipeSlaveEqControl
RxEqEval			
	Input	rx_eq_eval_0 / attached_rx_eq_eval_0	PipeSerdesEqControl / PipeSlaveControl
InvalidRequest			
	Input	invalid_request_0 / attached_invalid_request_0	PipeSerdesEqControl / PipeSlaveEqControl
TxDeemph[17:0]			
	Input	tx_deemph_0 / attached_tx_deemph_0	PipeSerdesEqControl / PipeSlaveEqControl
FS[5:0]			
	Input	fs_0 / attached_fs_0	PipeSerdesEqControl / No connection

Table D-2 PIPE 4.3 Equalization Signals Not Used (Continued)

Name	Spipe Dir.	Lane 0 Mpipe / Spipe Name	Mpipe / Spipe Task
LF[5:0]			
	Input	lf_0 / attached_lf_0	PipeSerdesEqControl / No connection
RxPresetHint[2:0]			
	Input	rx_preset_hint_0 / attached_rx_preset_hint_0	PipeSerdesEqControl / PipeSlaveEqControl
LinkEvaluationFeedbackFigureMerit[7:0]			
	Output	link_eval_feedback_figure_of_merit_0 / attached_link_eval_feedback_figure_of_merit_0	PipeSerdesEqControl / PipeSlaveEqControl
LinkEvaluationFeedbackDirectionChange [5:0]			
	Output	link_eval_feedback_direction_change_0 / attached_link_eval_feedback_direction_change_0	PipeSerdesEqControl / PipeSlaveEqControl
RxEqInProgress			
	Input	rx_eq_in_progress_0 / attached_rx_eq_in_progress_0	PipeSerdesEqControl / PipeSlaveControl

D.2 Parameters

The following table describes the parameters used to control the PIE-8 interface.

Table D-3 PIE-8 Parameters

Parameter Name	Settable VAR*	Default Value	Description
PIE8_MODE_EN			
	YES	0	Enable PIE-8 mode (PHY Interface Extensions Supporting 8GT/s PCIe): Enables the PIE-8 mode state machines (either as "MAC master" or "PHY Slave" based on "IS_PIE8_MODE_MASTER").
IS_PIE8_MODE_MASTER			
	YES	0	If a "1", enables the model to behave as the MAC master for the PIE-8 interface: driver of the MacDataEn/ MacData[5:0] interface signals; responder to the PhyDataEn/ PhyData[5:0] signals. If a "0", enables the model to behave as the PHY slave for the PIE-8 interface: receiver of the MacDataEn/ MacData[5:0] interface signals; driver of the PhyDataEn/ PhyData[5:0] signals.

Table D-3 PIE-8 Parameters (Continued)

Parameter Name	Settable VAR*	Default Value	Description
PIE8_PHY_DELAY_TO_TX_CMD_OUT_MIN			
	YES	1	If performing as a PHY slave for the PIE-8 interface, This is the minimum number of "PClk cycles that the PIE-8 slave state machine will wait in the PHY_RX_WAIT_TX_PHY_RESP state before asserting "PHYDataEn" and proceeding toward completion. Selection is random between PIE8_PHY_DELAY_TO_TX_CMD_OUT_MIN_VAR and PIE8_PHY_DELAY_TO_TX_CMD_OUT_MAX_VAR.
PIE8_PHY_DELAY_TO_TX_CMD_OUT_MAX			
	YES	25	If performing as a PHY slave for the PIE-8 interface, This is the maximum number of "PClk cycles that the PIE-8 slave state machine will wait in the PHY_RX_WAIT_TX_PHY_RESP state before asserting "PHYDataEn" and proceeding toward completion. Selection is random between PIE8_PHY_DELAY_TO_TX_CMD_OUT_MIN_VAR and PIE8_PHY_DELAY_TO_TX_CMD_OUT_MAX_VAR.
PIE8_ENABLE_EQUALIZATION_CHECKS			
	YES	0	When set to a "1", it enables the PIE-8 checks in the "Pie8PhyStateMachine".
PIE8_EQUALIZATION_CHECK_VECTOR			
	YES	3'b111	The following checks are only performed if the "PIE8_ENABLE_EQUALIZATION_CHECKS_VAR" is a "1": • Bit[0]: when a "1", the received "preset" is compared against the appropriate "upstream_preset_reg", "upstream_preset_reg_16g", "downstream_preset_reg" or "downstream_preset_reg_16g" based on current "data_rate" and Lane direction for the "Set Initial Presets" MAC command. • Bit[1] when a "1", the received "preset_hint" is compared against the appropriate "upstream_preset_hint_reg", "upstream_preset_hint _reg_16g", "downstream_preset_hint _reg" or "downstream_preset_hint_reg_16g"based on current "data_rate" and Lane direction for the "Set Initial Presets" MAC command. • Bit[2] when a "1", the received "preset" is checked to see that it has a "valid" table entry in the "upstream_tx_preset_coefficient_mapping_table", "uThis is the minimum number of PClk cycles that the PIE-8 pstream_tx_preset_coefficient_mapping_table_16g", "downstream_tx_preset_coefficient_mapping_table" or "downstream_tx_preset_coefficient_mapping_table_16g" based on current "data_rate" and Lane direction for the "Request Local Preset" MAC command.
PIE8_MAX_MAC_WAIT_DELAY_TO_PHY_DATAEN_TIMEOUT			
	YES	10000	The maximum time (in nS) that a Lane's Pie8MacStateMachine will wait in its "TX_WAIT_RX_PHY_RESP" state for the "PHYDataEn" signal to be received (signaling returned data from that Lane's corresponding "Pie8PhyStateMachine"). Timeout and return to "Idle" when expiring.

D.3 PHY PIE-8 ASCII Signals

The following table lists the ASCII signals on the PIE-8 PHY.

Table D-4 PHY PIE-8 ASCII Signals

Signal Name	Description
ascii_pie8_lane<n>_mac_state	<n> = 0 to 31. Current state of Lane <n>’s MAC PIE-8 Master state machine
ascii_pie8_lane<n>_phy_state	<n> = 0 to 31. Current state of Lane <n>'s MAC PIE-8 Slave state machine

D.4 PHY PIE-8 Internal Signals

The following signals may be helpful in debugging DUT issues (only one of these machines will be active in a given "root" or "endpoint" model). They are per lane and for both the MAC and PHY.

Table D-5 MAC Internal PIE-8 "per Lane" Signals

Signal Name	Description
pie8_mac_state	Current state of a Lane's MAC PIE-8 Master state machine
last_pie8_mac_state	Previous state of a Lane's MAC PIE-8 Master state machine
pie8_cycle_en	"Per bit" start of a Lane's MAC PIE-8 Master state machine
pie8_command	Current active command of a Lane's MAC PIE-8 Master state machine (if pie8_cycle_en[lane_num]" is a "1").
pie8_command_done	Last command of a Lane's MAC PIE-8 Master state machine has completed.
pie8_num_tx_data	Number of data cycles to be transmitted by a Lane's MAC PIE-8 Master state machine. Loaded when command starts and pre-decremented as data is transmitted.
pie8_phy_rx_control	First datum received by a Lane's MAC PIE-8 Master state machine when the "PhyDataEn" for that lane is first asserted. Valid only when pie8_cycle_en[lane_num]" is a "1".
pie8_exp_num_rx_data	Number of data cycles to be received by a Lane's MAC PIE-8 Master state machine. Will start out as greater than 0 for MAC commands that expect a PHY response. Loaded when "PhyDataEn" for that lane is first asserted and pre-decremented as data is received. Valid only when pie8_cycle_en[lane_num]" is a "1".

Table D-6 PHY Internal PIE-8 "per Lane" Signals

Signal Name	Description
pie8_phy_state	Current state of a Lane's PHY PIE-8 Slave state machine
last_pie8_phy_state	Previous state of a Lane's PHY PIE-8 Slave state machine

Table D-6 PHY Internal PIE-8 "per Lane" Signals (Continued)

Signal Name	Description
pie8_num_tx_data	Number of data cycles to be transmitted by a Lane's MAC PIE-8 Slave state machine. Loaded when command starts and pre-decremented as data is transmitted.
pie8_exp_num_rx_data	Number of data cycles to be received by a Lane's PHY PIE-8 Slave state machine from the MAC. Will start out as greater than 0 for MAC commands that expect a PHY response. Loaded when "MACDataEn" for that lane is first asserted and pre-decremented as data is received.

D.5 PIE-8 Protocol Check "MSGCODEs"

The following table list the "MSGCODEs" associated with PIE-8 protocol and data checking.

Table D-7 PIE-8 MSGCODES

Signal Name	Description
MSGCODE_PCIESVC_PIE8_PHY_RX_PRESET_ MISCOMPARE	
	The "Pie8PhyStateMachine" checks that the "preset" it received from the MAC in the "Set Initial Presets" command is the one expected based on what was saved by the VIP in its respective "upstream_preset_reg" or "downstream_preset_reg" at the 8.0 GT/s data rate.
MSGCODE_PCIESVC_PIE8_PHY_RX_PRESET_ MISCOMPARE_16G	
	The "Pie8PhyStateMachine" checks that the "preset" it received from the MAC in the "Set Initial Presets" command is the one expected based on what was saved by the VIP in its respective "upstream_preset_reg_16g" or "downstream_preset_reg_16g" at the 16.0 GT/s data rate.
MSGCODE_PCIESVC_PIE8_PHY_RX_PRESET_ HINT_MISCOMPARE	
	The "Pie8PhyStateMachine" checks that the "preset hint" it received from the MAC in the "Set Initial Presets" command is the one expected based on what was saved by the VIP in its respective "upstream_preset_hint_reg" or "downstream_preset_hint_reg" at the 8.0 GT/s data rate.
MSGCODE_PCIESVC_PIE8_PHY_RX_PRESET_ HINT_MISCOMPARE_16G	
	The "Pie8PhyStateMachine" checks that the "preset hint" it received from the MAC in the "Set Initial Presets" command is the one expected based on what was saved by the VIP in its respective "upstream_preset_hint_reg_16g" or "downstream_preset_hint_reg_16g" at the 16.0 GT/s data rate.
MSGCODE_PCIESVC_PIE8_PHY_RX_PRESET_ TABLE_ENTRY_INVALID	
	The "Pie8PhyStateMachine" checks that the "preset" it received from the MAC in the "Request Local Preset" command has a valid entry (checks the "VALID" bit - not the coefficient rules) in its respective "upstream_tx_preset_coefficient_mapping_table" or "downstream_tx_preset_coefficient_mapping_table" at the 8.0 GT/s data rate.
MSGCODE_PCIESVC_PIE8_PHY_RX_PRESET_ TABLE_ ENTRY_INVALID_16G	
	The "Pie8PhyStateMachine" checks that the "preset" it received from the MAC in the "Request Local Preset" command has a valid entry (checks the "VALID" bit - not the coefficient rules) in its respective "upstream_tx_preset_coefficient_mapping_table_16g" or "downstream_tx_preset_coefficient_mapping_table_16g" at the 16.0 GT/s data rate.
MSGCODE_PCIESVC_PIE8_PHY_RX_MAC_DATA_ LESS_THAN_EXP	
	The "Pie8PhyStateMachine" checks in its "PHY_RX_DATA" state that it receives the expected number of datums from the MAC for any command that receives data (more than just the "control" byte). If "MACDataEn" de-asserts before the pie8_exp_num_rx_data" for the Lane reaches "0", this check will fire.

Table D-7 PIE-8 MSGCODES (Continued)

Signal Name	Description
MSGCODE_PCIESVC_PIE8_PHY_RX_INVALID_ MAC_DATA_EN	
	The "Pie8PhyStateMachine" checks in its "PHY_RX_WAIT_TX_PHY_RESP" state to see if the "MACDataEn" is driven to a "1" again. If it is, this check will fire.
MSGCODE_PCIESVC_PIE8_PHY_RX_ UNSUPPORTED_MAC_CONTROL	
	The "Pie8PhyStateMachine" checks in its "PHY_RX_IDLE" state whether the "control" value sent when "MACDataEn" is driven to a "1" initially is a valid "command". If it is a "reserved" or "unsupported" command (such as "Set Lane Number"), this check will fire.
MSGCODE_PCIESVC_PIE8_PHY_MAC_DATA_ EN_RESTARTED	
	The "Pie8PhyStateMachine" checks in its "PHY_TX_CMD_OUT", "PHY_TX_DATA" and "PHY_WAIT_EVAL_RESP" states to see if the "MACDataEn" is driven to a "1" again. If it is, this check will fire.
MSGCODE_PCIESVC_PIE8_MAC_ UNSUPPORTED_COMMAND	
	The "Pie8MacStateMachine" checks in its "MAC_TX_IDLE" state whether its "per Lane" command ("pie8_command[lane_num]") that it received from the LTSSM when its "pie8_command[lane_num]" bit was asserted is a valid PIE-8 MAC "Information Transfer". If it is not a valid MAC "Information Transfer", this check will fire and the state machine will remain in the "MAC_TX_IDLE" state, clearing the offending command.
MSGCODE_PCIESVC_PIE8_MAC_RX_DATA_ LESS_THAN_EXP	
	The "Pie8MacStateMachine" checks in its "MAC_RX_DATA" state whether its "per Lane" expected number of datums for the currently active command (pie8_exp_num_rx_data[lane_num]) have been received (reached "0") before the "PHYDataEn" is de-asserted. If "PHYDataEn" is de-asserted before this value reaches "0", this check will fire. This means that the PHY did not send all the data required for the command sent to it.
MSGCODE_PCIESVC_PIE8_MAC_RX_DATA_ MORE_THAN_EXP	
	The "Pie8MacStateMachine" checks in its "MAC_WAIT_PHY_DATA_EN_DROP" state that the "PHYDataEn" is de-asserted. If the "PHYDataEn" remains asserted, this check will fire. This means that the PHY is sending too much data for the command sent to it.
MSGCODE_PCIESVC_PIE8_MAC_WAIT_PHY_ DATAEN_TIMEOUT	
	The "Pie8MacStateMachine" checks in its "MAC_TX_WAIT_RX_PHY_RESP" state that the "PHYDataEn" is asserted before the timeout defined by the "PIE9_MAX_MAC_WAIT_DELAY_TO_PHY_DATAEN_TIMEOUT_VAR" (in nS). If "PHYDataEn" is not asserted within this timeout value, the MAC state machine will fire this check, clear the current command and return to its "MAC_TX_IDLE" state. This would be the result of the PHY not responding in time to the MAC "Information Transfer" command.

D.6 PIE-8 Error Injection "EIs"

The following table lists "Error Injections" associated with PIE-8 protocol. There are injections for the MAC "master" state machine, and for the PHY "slave" state machine. The controls can be used to weight types or errors.

Table D-8 PIE-8 MAC State Machine Error Injections

Name	Description	DUT Action
EI_TX_PIE8_MAC_SM_PERCENTAGE		
	Percentage probability of a MAC State Machine type EI occurring	
EI_TX_PIE8_MAC_SM_LANE_MASK		
	A bitwise enable mask for only applying TX_PIE8_MAC_SM EIs to certain lanes. Default value is 32'hFFFF_FFFF (all lanes enabled).	
EI_TX_PIE8_MAC_SM_UNDEFINED_CONTROL_CODE_WEIGHT		
	Changes the control code issued when MACDataEn is first asserted to an undefined value. The actual value is selected between EI_TX_PIE8_MAC_SM_UNDEFINED_CONTROL_CODE_VALUE_MIN (6) and EI_TX_PIE8_MAC_SM_UNDEFINED_CONTROL_CODE_VALUE_MAX (63)	DUT should not detect a valid control code and should not respond with any PHYDataEn.

Table D-9 PIE-8 PHY State Machine Error Injections

Name	Description	DUT Action
EI_TX_PIE8_PHY_SM_PERCENTAGE		
	Percentage probability of a PHY State Machine type EI occurring.	
EI_TX_PIE8_PHY_SM_LANE_MASK		
	A bitwise enable mask for only applying TX_PIE8_PHY_SM EIs to certain lanes. Default value is 32'hFFFF_FFFF (all lanes enabled).	
EI_TX_PIE8_PHY_SM_UNDEFINED_CONTROL_CODE_WEIGHT		
	Changes the control code issued when PHYDataEn is first asserted to an undefined value. The actual value is selected between EI_TX_PIE8_PHY_SM_UNDEFINED_CONTROL_CODE_VALUE_MIN (7) and EI_TX_PIE8_PHY_SM_UNDEFINED_CONTROL_CODE_VALUE_MAX (63).	DUT should not detect a valid control code when PHYDataEn is asserted in response to its MACDataEn.

E
PCIe Verilog Testbench Example

This chapter describes how to install and run a Verilog PCIe example. Topics covered are:

- "Testbench Example Description"
- "PCIe VIP Example Files"
- "Installing and Running Examples"

E.1 Testbench Example Description

The Verilog PCIe VIP example contains two VIPs connected together. One instance, named root0, is configured as the upstream device and the other instance, named endpoint0, is configured as the downstream device. In the Example/test_basic.v file, a reset is asserted at the beginning of simulation. The transaction log file is also initiated, using the msglog_control task.

After reset is deasserted, the VARs are set and the two instances are configured. For example, the SetSupportedSpeeds, SetLinkWidth, SetRequesterID, link_enable, and others are set for each instance.

Once the VIP is ready, you can issue commands from the application layers. Check if the VIP is ready by using these commands:

```
wait (top.root0.port0.pl0_phy_ready == 1); // phy ready
wait (top.root0.port0.dl0_status == 1); // DL ready
```

Make sure to set up memory for your buffers. In the Verilog example, the root0 device sends a configuration read using QueueCfgRead. The testbench than sets up to do a memory write using QueueMemWrite. As part of setting up the memory write, the data, address, and data length are randomized. A QueueMemRead is issued, and the testbench waits for the completion of the memory read request.

Another memory write and read are issued. The tx_ei_code is set up to inject an lcrc error. It is commented out, but can be uncommented so that the error is injected. After the memory write and read tasks, the testbench waits for the link to become idle.

```
top.root0.port0.dl0.IsDataLinkIdle(status);
top.endpoint0.port0.dl0.IsDataLinkIdle(tmp_status);
status = status & tmp_status;
top.root0.port0.tl0.IsTransactionLayerIdle(tmp_status);
status = status & tmp_status;
top.endpoint0.port0.tl0.IsTransactionLayerIdle(tmp_status);
```

```
        status = status & tmp_status;
```

Once the link has become idle, check for statistics of the simulation using these tasks:

> top.root0.port0.dl0.DisplayStats;
>
> top.root0.port0.tl0.DisplayStats;

The simulation log file will be placed in the output directory. For a successful simulation run, the simulation log file should not show any warnings or errors.

E.2 PCIe VIP Example Files

The following files are provided for use in Verilog PCIe VIP examples:

- PCIE/Verilog/Example/test_basic.v

  Top-level connection between two PCIe VIPs, root complex and endpoint.
- config_root2endpoint.v

  Configuration of the two VIPs. The default is x4 pipe interface.
- config_root2endpoint_8g.v

  Configuration of the two VIPs for gen3. The default is x4 pipe interface.
- config_root2endpoint_x8.v

  Configuration of the two VIPs. The default is x8 pipe interface.
- config_root2endpoint_x8_8g.v

  Configuration of the two VIPs for gen3. The default is x8 pipe interface.
- config_root2endpoint_x32.v

  Configuration of the two VIPs. The default is x32 pipe interface.
- config_root2endpoint_x32_8g.v

  Configuration of the two VIPs for gen3. The default is x32 pipe interface.
- test_basic_uvm.sv

  UVM example of two PCIE VIPs connected, root complex and endpoint)
- test_basic_sv.sv

  VMM example.
- test_mem_target_uvm.sv

  Memory target UVM example.
- test_uvm_nvme.sv

  NVME UVM example.
- uvm_env_root2endpoint.sv

  uvm_env that goes with the config_root2endpoint.v top.

The following test_*.fl files are used for including files and defining interfaces in the compilation:

- test_basic.fl

  File list for the basic Verilog example.
- test_basic_nvme.fl

File list for the NVME example.

- test_basic_sv.fl

  File list for the SystemVerilog example.
- test_basic_uvm_(nc|questa|vcs).fl

  File list for the basic UVM example for Synopsys, Cadence, and Mentor simulators.
- test_mem_target_uvm_(nc|questa|vcs).fl

  File list for memory target example for Synopsys, Cadence, and Mentor simulators.
- test_uvm_nvme_(nc|questa|vcs).fl

  File list for the nvme uvm example for Synopsys, Cadence, and Mentor simulators.
- Makefile

  Utilities to run the example test.
- README.uvm

  Descriptions of the example testbenches and instructions for running them.

E.3 Installing and Running Examples

E.3.1 Download the PCIe VIP tar File

Follow the steps below to download the PCIe VIP from SolvNet.

1. Create a directory in which to install the PCIe VIP files. That directory is called *installation_dir* in this procedure.
2. Go to this SolvNet page:

   https://solvnet.synopsys.com/ESTCryptic?agreement=Y&material_id=44803-EST&version=%2Fauth%2Feio_vip_vG-2012.09-3.auth

   You will see a list of products titled “My Product Releases”.
3. Select ExpertIO VIP from the list under My Product Releases.

   You will see a list of version numbers, such as H-2012.12-07, titled “ExpertIO VIP”.
4. Select the latest release number from the list under ExpertIO VIP.

   You will see a page titled “Downloads” with “Download Details: ExpertIO VIP: *version*” on it, followed by a Download Here button.
5. Click the Download Here button.

   You will see an Electronic Software Transfer page with a button labeled “YES, I AGREE TO THE ABOVE TERMS” on it.
6. If you agree to the download terms, click the YES, I AGREE TO THE ABOVE TERMS button.

   You will see a page with a list of VIP tar files on it witha Download button for each file.
7. Download one of the following tar files, where 1_1_*x* is the latest PCIe VIP release number:

   expertio_PCIE_1_1_*x*_V.tar if your simulator will be from Synopsys

   expertio_PCIE_1_1_*x*_N.tar if your simulator will be from Cadence

   expertio_PCIE_1_1_*x*_M.tar if your simulator is from Mentor Graphics

8. Save the tar file in the *installation_dir* that you created in Step 1.
9. Untar the PCIe VIP file in your *installation_dir*.

E.3.2 Download the DesignWare Licensing and Shared Library Files

Get the DesignWare licensing files. Information about licensing is contained in this SolvNet article:

https://solvnet.synopsys.com/retrieve/037139.html

A shared library in the Synopsys DesignWare suite is needed to interface between the SVC and the FlexLM mechanism. If you have not already installed it, follow these steps to install it:

1. From the UNIX prompt, create a directory named dw_home, in which to install the DesignWare shared library files. That directory is called *path*/dw_home in this procedure.
2. Log into the SolvNet Download Center at the following location:

   https://solvnet.synopsys.com/DownloadCenter

   You will see a Downloads page with a list titled “My Product Releases”.
3. Click "Discovery Verification IP".
4. In the "Discovery Verification IP" list, click F-2011.12.

   You will see a page with “Download Details: Discovery Verification IP: F-2011.12” and a “Download Here” button on it.
5. Click the Download Here button.

   You will see an “Electronic Software Transfer” page.
6. Click YES, I AGREE TO THE ABOVE TERMS to accept the download terms.

   You will see a list of files with a Download button for each file.
7. Click the Download button for the dw_vip_common_5.95a.run file.
8. Save the file in a UNIX directory of your choice.
9. From the UNIX prompt in that directory, execute the following command to install the DesignWare shared library files:

   ```
   $ dw_vip_common_5.95a.run --dir path/dw_home
   ```
10. You will be prompted to confirm that you want to install the libraries in the specified directory. Press y or Enter to start the installation.

    When the installation is finished you will see the message “dwh_install: Done.”

E.3.3 Set Up Your Environment

Set the following environment variables:

1. DESIGNWARE_HOME

   Set the DESIGNWARE_HOME environment variable to the path to the dw_home directory you created in “Download the DesignWare Licensing and Shared Library Files”

   ```
   setenv DESIGNWARE_HOME path/dw_home
   ```
2. UVM_HOME

   Set the UVM_HOME environment variable to the full hierarchical path of the UVM source directory. UVM version 1.0p1 or later is required.

```
setenv UVM_HOME /usr/local/uvm/uvm-1.0p1.
```

Examples

For a Synopsys environment:

```
setenv UVM_HOME /synopsys_home/etc/uvm-1.1
```

For a Cadence environment:

```
setenv UVM_HOME /cadence_home/tools/uvm-1.1
```

For a Mentor environment:

```
setenv UVM_HOME /mentor_home/modeltech/Verilog_src/uvm-1.0p1
```

3. ARCH

   Set the ARCH environment variable to your architecture. (In a Cadence environment, the cds_plat script prints out the value for the architecture.

   Supported values for ARCH are:

   lnx86 (Linux on x86)

   sun4v (sun-solaris)

   setenv ARCH lnx86

   For example:

   ```
   setenv ARCH lnx86
   ```

4. SVC_SIM

   Set the SVC_SIM environment variable to one of the following:

   VCS (Synopsys VCS)

   NC (Cadence NC-Verilog)

   MT (Mentor Graphics ModelTech)

   For example:

   ```
   setenv SVC_SIM VCS
   ```

5. Set your simulator environment variable.

   For Synopsys VCS:

   - VCS_HOME – Set to point to your local Synopsys installation tree.

     Default: none

   - Command line options – The following typically are required command line options:

     +v2k – Verilog 2K

     -P *installation_dir*/*product*/Verilog/Util/pli.tab – PLI table

     installation_dir/*product*/Verilog/Util/Msglog/msglog.o – Loads the msglog PLI object

     +incdir+*installation_dir*/*product*/Verilog – Sets the include path

     NOTE: For all simulators running on 64-bit Linux machines, when the 32-bit NC-Verilog is being used, slightly modified compilation and linking steps are required to generate a 32-bit PLI library. In a Synopsys environment, you must set the environment variable SVC_USE_64BIT_LNX to 1 (that is, `setenv SVC_USE_64BIT_LNX 1`). In this case, the VIP does not run in 64-bit mode, but uses 32-bit emulation instead.

Whenever changing the value of this variable, you must run `make clean` from the top (*product*/ Verilog) directory to remove old object files.

For Cadence NC-Verilog:

- CDS_INST_DIR – Set the CDS_INST DIR environment variable to point to your local Cadence installation tree.

  Defaults: none

  CDS_INST_DIR is typically set to /usr/local/cds or /usr/local/cadence.

- LD_LIBRARY_PATH – Set the LD_LIBRARY_PATH environment variable to the path to your shared libraries. Both the Cadence shared libraries and those that are under the Verilog sub-directory in the SVC installation directory are needed.

  Default: None

  LD_LIBRARY_PATH is typically set to $CDS_INST_DIR/tools/lib:*installation_dir*/*product*/Verilog (where *product* is one of PCIE, SAS, SATA, FC, ETH, ...)

- Command line options – The following typically are required command line options:

  +loadpli1=dyn_libpli.so:svc_msglog.svc_msglog – Loads the msglog PLI

  +incdir+*installation_dir*/*product*/Verilog – Sets the include path

For ModelTech/Mentor MTI or Questa:

- MT_INST_DIR - Set to point to your local MTI installation tree.

  Default: None

- Command line options – The following typically are required command line options:

  -incr – Incremental compilation

  -R -c -wlf vsim.wlf - – Simulation arguments (the trailing "-" is required)

  +incdir+*installation_dir*/*product*/Verilog – Sets the include path

6. Set the SVC installation path to the Verilog directory. The files in the Example directory will use this variable:

```
setenv EXPERTIO_PCIESVC_INSTALL_PATH
    /installation_dir/expertio_PCIE_1_1_x_V/PCIE/Verilog
```

7. Set up your path, LD_LIBRARY_PATH and LM_LICENSE_FILE

   - PATH – Set to invoke common development executables such as make, gcc, and your simulator.
   - LD_LIBRARY_PATH – Set to your default path to shared libraries. We attempt to set this automatically for you in the scripts provided with the VIP.
   - LM_LICENSE_FILE – Point to your license file or license server. This is specific to your environment. Examples:

     /usr/local/flexlm/licenses/license.dat

     or

     5280@your_license_server

E.3.4 Run the Example

1. In the PCIE/Verilog directory, run the `make` command to build the dyn_libpli.so library file.

2. Go to the Example directory and run `make`.

   The following `make` command examples are for the Linux 64-bit platform.

   - Verilog example:

     ```
     make test_basic VC_ARGS="$DESIGNWARE_HOME/vip/common/latest/C/lib/amd64/
     VipCommonNtb.so +define+WAVEFORMS +msglog_transaction_file=output/run.xlog"
     ```

   - UVM example:

     ```
     make test_basic_uvm_vcs VC_ARGS="$DESIGNWARE_HOME/vip/common/latest/C/lib/
     amd64/VipCommonNtb.so +define+WAVEFORMS +msglog_transaction_file=output/
     run.xlog"
     ```

F PCIe UVM Testbench Example

The PCIe UVM SVC example contains two VIP components connected back to back. One component is configured as the upstream device, named root0, and the other instance is configured as the downstream device, named endpoint0.

The test_basic_uvm.sv file in the Example directory has all of the test-specific functionality defined within a UVM test class. The config_root2endpoint.v file, also in the Example directory, provides the testbench environment class.

The root0 and endpoint0 UVM component instantiations are done in the UVM environment class. The UVM test uses an instance of the environment to apply its test-specific scenarios.

The test asserts the reset signal for a 100ns period. After the de-assertion of the reset, the two components are configured to run in a specific mode of operation.

The following configuration steps are done in the example test:

1. Configure the Virtual Channel (VC) configuration and set up flow control credit.

   The pciesvc_tl_vc_config_item class is used to configure the VCs and initialize the credit counters of the UVM component. This configuration is applied on the Transaction Layer of the component.

```
begin
   pciesvc_tl_vc_config_item root0_vc_config_item;

   …

   // Enabling VC0
   root0_vc_config_item.vc.rand_mode(0);
   root0_vc_config_item.vc = 0;
   root0_vc_config_item.enable.rand_mode(0);
   root0_vc_config_item.enable = 1;

   // Map Traffic Class (TC=0) to VC0
   root0_vc_config_item.tcs_to_map_to_vc.rand_mode(0 );
   root0_vc_config_item.tcs_to_map_to_vc = 8'b0000_0001;
```

```
      rand_result = root0_vc_config_item.randomize() with {
         // Setting initial credits
         p_hdr_credits inside { [128:150] };
         p_data_credits inside { [512:600] };
         np_hdr_credits inside { [128:150] };
         np_data_credits inside { [512:600] };
         cpl_hdr_credits inside { [128:150] };
         cpl_data_credits inside { [512:600] };
      };

      // Apply the configuration on the vf_config_agent's sequencer
      env.root0.port[0].tl[0].vc_config_agent.sequencer.execute_item(
         root0_vc_config_item);
   end
```

The same setting is applied on the endpoint0 component instance.

2. Set up the link width and link speed.

   The pciesvc_phy_set_link_width_item and pciesvc_phy_set_supported_speeds_item are used to configure the targeted link width and link speed in the example test. This configuration is applied to the Phy Layer of the component instance.

```
begin
   pciesvc_phy_set_link_width_item set_link_width;
   pciesvc_phy_set_supported_speeds_item set_supported_speeds;
   …

   // Setting a X1 lane configuration
   set_link_width.link_width = LINK_WIDTH_X1;
   env.root0.port[0].pl[0].control_agent.sequencer.execute_item(set_link_width);

   // Setting a 2.5G link speed
   set_supported_speeds.speeds = SPEED_2_5G;
   env.root0.port[0].pl[0].control_agent.sequencer.execute_item(
      set_supported_speeds);
end
```

   The same setting is applied on the component instance endpoint0.

3. Enable the Data Link Layer and the Phy Layer.

   The pciesvc_phy_set_phy_enable_item and pciesvc_dl_set_link_enable_item are used to enable the Phy Layer and the Data Link Layer respectively. The enable on the Phy Layer causes the LTSSM to transition out of the disabled state. The enable on the Data Link Layer enables the Data Link Layer.

```
begin
   pciesvc_phy_set_phy_enable_item root0_set_phy_enable_item;

   root0_set_phy_enable_item = new("root0_set_phy_enable_item");
   root0_set_phy_enable_item.enable = 1;
```

```
      // Use the PL’s control agent’s sequencer to enable the PL
      env.root0.port[0].pl[0].control_agent.sequencer.execute_item(
         root0_set_phy_enable_item);
   end

   begin
      pciesvc_dl_set_link_enable_item root0_set_link_enable_item;
      root0_set_link_enable_item = new("root0_set_link_enable_item");
      root0_set_link_enable_item.enable = 1;

      // Use the DL’s control agent’s sequencer to enable the DL
      env.root0.port[0].dl[0].control_agent.sequencer.execute_item(
         root0_set_link_enable_item);
   end
```

The same setting is applied on the component instance endpoint0.

After applying all of the above settings, the test waits for the Phy Layer and Data Link Layer to reach a ready state before using the driver application to generate TLP traffic.

Once the Physical Layer and the Data Link Layer are ready, the example test applies some test-specific settings to the VAR parameters of the driver application to influence the generation of the TLPs. The test then executes a sequence that applies a CfgRd0 TLP on the root0 component instance. Upon receiving a successful completion for the CfgRd0 TLP, it forks off a memory TLP sequence on both root0 and endpoint0 instances.

The sequences defined in the example are standard UVM sequences (extending from the uvm_sequence class) that generate sequence items of type pciesvc_driver_transaction_item. The body() forms the core of the class definition, scheduling items and transactions for execution. The test defines and executes two sequences:

1. cfg_transaction_sequence – Generates a configuration read TLP.
2. multi_transaction_sequence – Generates a sequence of memory write followed by a read from the same address.

At the end of the test, the check_phase() is defined to query the number of errors and warnings that were encountered in simulation. The simulation log file is placed in the output directory. It should not show any warnings or errors for a successful simulation run.

G
Implemented ECNs

Engineering Change Notices (ECNs) that have been implemented in the PCIe VIP are listed in Table G-1.

Table G-1 ECNs implemented in the PCIe VIP

ECN	Spec Version	Comments
L1 sub-states	3	Supported
OBFF	3	Supported
LTR	3	Supported
SR-IOV	2.1	Supported
ARI Capability	2.1	Supported
DPC	3	Supported
TLP Prefixes	2.1	Supported
FLR	2.1	Supported EP
ASPM Optionality	2.1	Supported
TLP Processing Hints	2.1	Supported
Extended Tag Enable	2.1	Supported
ID Based Ordering	2.1	Supported
Atomic Operations	2.1	Supported
ARI Capability	2.1	Supported
Address Translation Serv	2.1	Supported

H
Reporting Problems

H.1 Introduction

This chapter outlines the process for working through and reporting transactor issues encountered in the field. It describes the data you must submit when a problem is initially reported to Synopsys. After a review of the initial information, Synopsys may decide to request adjustments to the information being requested, which is the focus of the next section.

H.2 Initial Customer Information

Follow these steps when you call the Synopsys Support Center:

1. Before you contact technical support, be prepared to provide the following:
   - A description of the issue under investigation
   - A description of your verification environment
2. Create a value change dump (VCD) file
3. Generate a log file for the simulation

   Providing this information will help ensure accurate diagnosis of the problem.

   If the requested information is not sufficient to determine the cause of your issue, the Synopsys Support Center may request a simple testbench demonstrating the issue. This is the most effective way to debug issues, but if an example cannot be provided, Synopsys may request additional information that could include regenerating the log file using different settings.

   If your testbench initializes the random seed (see 'srandom()' in the VCS manual) in the host simulator, then a seed that can be used to demonstrate the problem must be included in the testbench or provided as information for executing the testbench.